Australian Government
Department of Defence
Defence Science and
Technology Organisation

Globally Optimal Path Planning with Anisotropic Running Costs

Jason R. Looker

Air Operations Division

Defence Science and Technology Organisation

DSTO–TR–2815

ABSTRACT

There are many diverse numerical methods that can be applied to solving path planning problems, however most of these are either not valid or impractical for solving anisotropic (direction-dependent) path planning problems. Ordered Upwind Methods (OUM) are a family of numerical methods for approximating the viscosity solution of static Hamilton-Jacobi-Bellman equations, and have been tailored to solve anisotropic optimal control problems.

There is little information in the literature regarding the implementation of OUM, and a wide range of computational techniques and meticulous algorithmic considerations are required to successfully implement OUM. A comprehensive, generic implementation of OUM is documented in this report, with the intention of minimising the technical barriers to employing OUM in real-world applications.

Published by

DSTO Defence Science and Technology Organisation
506 Lorimer St,
Fishermans Bend, Victoria 3207, Australia

Telephone: (03) 9626 7000
Facsimile: (03) 9626 7999

AR No. AR 015–556
March, 2013

APPROVED FOR PUBLIC RELEASE

Globally Optimal Path Planning with Anisotropic Running Costs

Executive Summary

Path planning is the task of selecting a path through an environment for an entity to traverse such that a cost is minimised. Path planning has numerous applications, for example, in robotics, computer game development, and controlling unmanned vehicles. This report emerged from a study of path planning for military aircraft conducting missions in hostile environments.

In this report, an entity is considered to follow a path in Euclidean space and incur a strictly positive running cost at each instant in time. The (total) cost is the integral of the running cost over the path, and the aim is to steer the entity through its environment such that this cost is minimised.

The running cost is considered to be anisotropic, that is, it depends on the position and velocity vector of the entity. Hence for anisotropic path planning problems, the cost depends on the location of the entity in its environment *and* how it arrived at that location. In applications, anisotropic running costs typically stem from heterogeneous environments, and cases where the orientation of the entity has a significant impact on the cost.

There are many diverse numerical methods that can be applied to solving path planning problems, however most of these are either not valid or impractical for solving anisotropic path planning problems. Ordered Upwind Methods (OUM) are a family of numerical methods for approximating the viscosity solution of static Hamilton-Jacobi-Bellman equations, and have been tailored to solve anisotropic optimal control problems. The focus of this report is on the control-theoretic OUM.

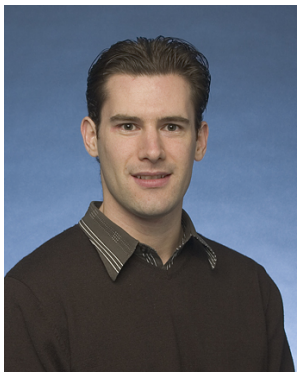
There is little information in the literature regarding the implementation of OUM, and a wide range of computational techniques and meticulous algorithmic considerations are required to successfully implement OUM. A comprehensive, generic implementation of OUM is documented in this report, with the intention of minimising the technical barriers to employing OUM in real-world applications.

The control-theoretic OUM has been employed to solve a number of simple path planning problems in this report, to provide figures with which the reader can compare output from their implementation of the OUM.

The application of OUM to the path planning of military aircraft traversing hostile environments is the subject of a future report.

THIS PAGE IS INTENTIONALLY BLANK

Author



Jason Looker
Air Operations Division

Dr Jason Looker joined DSTO in 2006 as an Operations Research Scientist after completing a BSc (Honours) and PhD in Mathematics at The University of Melbourne. He has undertaken operations analysis in support of Air Force Headquarters and AIR 9000 Phase 8 (Future Naval Aviation Combat System), and is leading a research project on the path planning of military aircraft through hostile environments.

THIS PAGE IS INTENTIONALLY BLANK

Contents

Appendices

Figures

THIS PAGE IS INTENTIONALLY BLANK

Glossary

DPP	Dynamic Programming Principle
HJB	Hamilton-Jacobi-Bellman
OUM	Ordered Upwind Method(s)

Notation

Vectors in real n-dimensional space are denoted by bold symbols, with the exception of the control $\boldsymbol{\alpha}$, which can be a scalar or an m-dimensional vector.

$\mathbb{R}^n$	real n-dimensional space
$\mathbf{x}$	a generic n-dimensional vector, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$
$\|\mathbf{x}\|$	magnitude of $\mathbf{x}$, $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}$
$\|\mathbf{x}\|_\infty$	infinity-norm, $\|\mathbf{x}\|_\infty = \max\{\|x_1\|, \|x_2\|, \ldots, \|x_n\|\}$
$\mathbf{x} \cdot \mathbf{y}$	scalar (dot) product, $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$
$\emptyset$	empty set
$\cap$	set intersection
$\cup$	set union
$\setminus$	set difference (relative complement)
$\subset$	subset
$\subseteq$	subset or equal
$\times$	Cartesian product
Ω	an open domain, $\Omega \subseteq \mathbb{R}^n$
t	time
s	arc length as a parameter
$\mathbf{y}_\mathbf{x}$	path with initial position at $\mathbf{x}$
$\dot{\mathbf{y}}_\mathbf{x}$	derivative of the path (velocity)
$\mathbf{f}$	dynamics of the path
f	speed of the path
$\boldsymbol{\alpha}$	control
$\boldsymbol{\alpha}^*$	(approximate) optimal control
$\mathcal{J}$	cost functional
$t_\mathbf{x}(\boldsymbol{\alpha})$	exit time associated with $\boldsymbol{\alpha}$ and $\mathbf{x}$ (earliest time for $\mathbf{y}_\mathbf{x}(t, \boldsymbol{\alpha})$ to hit the target set)
$\ell_\mathbf{x}(\boldsymbol{\alpha})$	arc length associated with $\boldsymbol{\alpha}$ and $\mathbf{x}$ (shortest arc length for $\mathbf{y}_\mathbf{x}(s, \boldsymbol{\alpha})$ to hit the target set)
$\overline{R}$	running cost per unit time
R	running cost per unit length
g	terminal cost
$\mathcal{T}$	closed target set where $\mathcal{T} \subset \Omega$
u	value function
inf	infimum (greatest lower bound)

$\mathcal{A}$	set of admissible controls
$\mathbf{e}_i$	ith standard basis vector in $\mathbb{R}^n$
∇	gradient vector differential operator, $\nabla = \sum_{i=1}^{n} \mathbf{e}_i \frac{\partial}{\partial z_i}$
h	triangulation diameter
X_h	triangulated mesh of diameter h
$\mathbf{x}_i$	a mesh point in X_h
Ω_h	discretisation of Ω
$\mathcal{T}_h$	discretisation of $\mathcal{T}$
Ω_h^3	$\Omega_h \times \Omega_h \times \Omega_h$
$V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$	tentative value function in the simplex $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$
V	tentative value function
U	numerical value function
$\mathrm{NF}(\mathbf{x})$	near front set as a function of $\mathbf{x} \in \Omega_h$
$\mathrm{NF}(i,j)$	near front set as a function of $(i,j) \in \Omega_h^{\mathbb{Z}}$
$\Upsilon(\mathbf{x})$	anisotropy function at $\mathbf{x}$
δ	Cartesian grid spacing
$\mathbb{Z}$	set of integers
(i,j)	integer mesh co-ordinate
$\mathbf{x}(i,j)$	mesh point in Ω_h with integer mesh co-ordinate (i,j)
$\Omega_h^{\mathbb{Z}}$	set of integer mesh co-ordinates
$\mathcal{T}_h^{\mathbb{Z}}$	set of integer mesh co-ordinates corresponding to $\mathcal{T}_h$
$\mathrm{int}(S)$	interior of a set S
$\mathbf{x}_{\mathcal{T}}$	target location in $\mathrm{int}(\mathcal{T})$
$\mathbf{e}_i^{\triangle}$	ith triangulation unit vector
$\rho_h(\Omega)$	computational radius
$N(i,j)$	neighbours of the mesh co-ordinate (i,j)
$N(S)$	neighbours of the set of mesh co-ordinates S, $N(S) \subseteq \Omega_h^{\mathbb{Z}} \setminus S$
$\overline{N}(i,j)$	neighbourhood of the mesh co-ordinate (i,j), $\overline{N}(i,j) = N(i,j) \cup \{(i,j)\}$
$P(\bar{i},\bar{j})$	set of all adjacent pairs in $\overline{N}(\bar{i},\bar{j})$
$\Delta(p)$	discriminant of the polynomial p, see Equation (26)
argmin	argument of the minimum of a function
$\overline{V}_{\mathbf{yz}}(\mathbf{x},\zeta)$	see Equation (28)
$T(i,j)$	the triplet $\{(i,j), V(i,j), \boldsymbol{\alpha}(i,j)\}$ for $(i,j) \in$ *Considered*
$T(i,j)$	the triplet $\{(i,j), U(i,j), \boldsymbol{\alpha}^*(i,j)\}$ for $(i,j) \in$ *Accepted*

THIS PAGE IS INTENTIONALLY BLANK

1 Introduction

Path planning is the task of selecting a path through an environment for an entity to traverse such that a cost is minimised. Path planning has numerous applications, for example, in robotics, computer game development, and controlling unmanned vehicles. This report emerged from a study of path planning for military aircraft conducting missions in hostile environments.

In this report, an entity is considered to follow a path in Euclidean space and incur a strictly positive running cost at each instant in time. The (total) cost is the integral of the running cost over the path, and the aim is to steer the entity through its environment such that this cost is minimised.

If there exists a path through the environment that incurs a lower cost than all other paths, then this path is globally optimal. Whereas if there exists a path through the environment that incurs a lower cost than all neighbouring paths (in some sense), then this path is locally optimal. A globally optimal path is also locally optimal, however a locally optimal path may not be globally optimal. Indeed, it is possible that a locally optimal path may represent a poor choice.

If the running cost only depends on the position of the entity, then the running cost is isotropic. Whereas the running cost is anisotropic if it also depends on the velocity vector of the entity. Hence for anisotropic path planning problems, the cost depends on the location of the entity in its environment *and* how it arrived at that location. In applications, anisotropic running costs typically stem from heterogeneous environments, and cases where the orientation of the entity has a significant impact on the cost.

This report details a comprehensive, generic implementation of a numerical method for approximating all globally optimal paths for anisotropic path planning problems. The application of this numerical method to the path planning of military aircraft traversing hostile environments is the subject of a future report.

Obstacle avoidance is a category of path planning where the environment consists of two distinct regions: one where the entity is free to move at a finite cost, and another where the entity is forbidden to enter. The literature on obstacle avoidance is vast, particularly in the fields of robotics and artificial intelligence, and methods to solve obstacle avoidance problems have been developed to exploit the allowed/forbidden nature of the environment to enable real-time computations. Since the path planning problem that motivated this report is not of an obstacle-avoidance type, the obstacle avoidance literature is not discussed here. For an introduction to obstacle avoidance, refer to the survey article by Hwang & Ahuja [1992] and the paper by LaValle & Kuffner, Jr. [2001]. Note that the numerical method that is the subject of this report can be used for obstacle avoidance problems.

There are many diverse approaches for solving path planning problems. These include methods from optimal control that are based on Pontryagin's Minimum Principle [Vian & Moore 1989], nonlinear programming [Betts 1998, Kabamba, Meerkov & Zeitz III 2006], and viscosity solutions of Hamilton-Jacobi-Bellman equations [Tsitsiklis 1995, Mitchell & Sastry 2003, Pêtrès et al. 2007]; the calculus of variations [Pachter & Hebert 2001, Novy, Jacques & Pachter 2002, Sidhu et al. 2006, Zabarankin, Uryasev & Murphey 2006]; math-

ematical programming [Kim & Hespanha 2003, Chaudhry, Misovec & D'Andrea 2004, Zabarankin, Uryasev & Murphey 2006, Muhandiramge, Boland & Wang 2009]; and heuristics based on Voronoi graphs and nonlinear programming [Dai & Cochran Jr. 2010], nonlinear trajectory generation [Milam, Mushambi & Murray 2000, Inanc et al. 2008], a system of virtual springs and masses [Bortoff 2000, Mercer & Sidhu 2007, Rowe, Sidhu & Mercer 2009], using simulation [Beard et al. 2002, McLain & Beard 2005], and evolutionary computation [Zheng et al. 2005]. Unfortunately, all of these approaches suffer from at least one of the following limitations:

- only locally optimal solutions are calculated;
- are heuristic and hence may not be locally optimal;
- are not valid for anisotropic running costs; or
- may not converge to the optimal path as the computational mesh is refined.

The final point primarily arises in graph-based methods, and has profound implications for the physical interpretation of the solution. For a discussion of this limitation, refer to the works of Sethian [1999*a*], Sethian [1999*b*], and Muhandiramge, Boland & Wang [2009]. A technique that does not suffer from any of these limitations is the control-theoretic Ordered Upwind Method [Sethian & Vladimirsky 2001, Vladimirsky 2001, Sethian & Vladimirsky 2003, Alton & Mitchell 2012].

Ordered Upwind Methods (OUM) are a family of numerical methods for approximating the viscosity solution of static Hamilton-Jacobi-Bellman equations. OUM closely resemble Dijkstra's seminal shortest-path algorithm for graphs [Dijkstra 1959], and bear scant resemblance to the usual finite difference methods for partial differential equations. The focus of this report is on control-theoretic OUM, which are based on a first order upwind approximation to the Dynamic Programming Principle of optimal control.[1]

The aim of this report is to document a generic implementation of the control-theoretic OUM,[2] as little detailed information is available in the literature. The OUM is introduced in Section 2, our implementation can be found in Section 3, and simple path planning examples have been included in Section 4 to provide the reader with points of comparison.

1.1 Optimal Control

The aspects of optimal control that are most relevant to OUM in a path planning context are outlined in this section. Refer to Bardi & Capuzzo-Dolcetta [2008, particularly Chapter IV] for a comprehensive exposition on optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations; Evans [1998] is also an excellent resource.

[1] Other OUM are based on upwind finite difference discretisations of the Hamilton-Jacobi-Bellman equation, however a proof of convergence for this approach for anisotropic control problems does not exist to the author's knowledge [Sethian & Vladimirsky 2003].

[2] "Control-theoretic OUM" and "OUM" will be used synonymously for the remainder of this report.

Let $\Omega \subseteq \mathbb{R}^n$ be an open domain. Initially the entity is at a position $\mathbf{x} \in \Omega$, then for each instant in time $t > 0$ the entity follows the path $\mathbf{y}_\mathbf{x}(t)$ according to the state equation

$$\dot{\mathbf{y}}_\mathbf{x}(t) = \mathbf{f}(\mathbf{y}_\mathbf{x}(t), \boldsymbol{\alpha}(t)) \quad (t > 0),$$

$$\mathbf{y}_\mathbf{x}(0) = \mathbf{x},$$

where $\dot{\mathbf{y}}_\mathbf{x}$ is the velocity, $\mathbf{f}$ is the (Lipschitz-continuous) dynamics, and $\boldsymbol{\alpha}$ denotes the control that steers the entity through its environment (see Section 1.2).

The cost is defined for each $\mathbf{x}$ and $\boldsymbol{\alpha}$ to be

$$\mathcal{J}(\mathbf{x}, \boldsymbol{\alpha}) = \int_0^{t_\mathbf{x}(\boldsymbol{\alpha})} \overline{R}(\mathbf{y}_\mathbf{x}(t), \boldsymbol{\alpha}(t))\, dt + g(\mathbf{y}_\mathbf{x}(t_\mathbf{x}(\boldsymbol{\alpha}))).$$

Here $t_\mathbf{x}(\boldsymbol{\alpha})$ is the exit time associated with $\boldsymbol{\alpha}$ and $\mathbf{x}$, which is the earliest time for $\mathbf{y}_\mathbf{x}(t, \boldsymbol{\alpha})$ to arrive at the closed target set $\mathcal{T} \subset \Omega$, $\overline{R}$ is the running cost per unit time, and g is the terminal cost.[3] Note that $t_\mathbf{x}(\boldsymbol{\alpha})$ needs to be determined: it is not an input.

The goal of path planning is to select a control from an admissible set $\mathcal{A}$ such that the cost is minimised. The value function $u(\mathbf{x})$ is defined to be the cost associated with a globally optimal path with initial position $\mathbf{x}$:

$$u(\mathbf{x}) = \inf_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{J}(\mathbf{x}, \boldsymbol{\alpha}).$$

The value function satisfies a functional equation referred to as the Dynamic Programming Principle (DPP):

$$u(\mathbf{x}) = \inf_{\boldsymbol{\alpha} \in \mathcal{A}} \left\{ \int_0^\tau \overline{R}(\mathbf{y}_\mathbf{x}(t), \boldsymbol{\alpha}(t))\, dt + u(\mathbf{y}_\mathbf{x}(\tau)) \right\} \quad (0 < \tau \leqslant t_\mathbf{x}(\boldsymbol{\alpha})).$$

If there exists a differentiable solution $u(\mathbf{x})$ of the DPP, then it can be shown that $u(\mathbf{x})$ also satisfies a nonlinear first order partial differential equation known as the (static) Hamilton-Jacobi-Bellman (HJB) equation:

$$\begin{cases} \min\limits_{\boldsymbol{\alpha} \in \mathcal{A}} \left\{ \mathbf{f}(\mathbf{x}, \boldsymbol{\alpha}) \cdot \nabla u(\mathbf{x}) + \overline{R}(\mathbf{x}, \boldsymbol{\alpha}) \right\} = 0 & \text{in } \Omega \setminus \mathcal{T}, \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \mathcal{T}. \end{cases}$$

However, a differentiable solution of the DPP may not exist, and solutions of the HJB equation are known to be non-unique and possess shock discontinuities.

A viscosity solution is a type of weak solution that has been developed to address these issues. In particular, viscosity solutions of the HJB equation are unique, continuous, equal to the differentiable solution at points of differentiability, and equal to the value function.

1.2 Admissible Controls

The control $\boldsymbol{\alpha}$ is considered to be a unit vector for the remainder of this report. In a path planning context, $\boldsymbol{\alpha}$ is typically the unit velocity of the entity and the dynamics are expressed as

$$\mathbf{f}(\mathbf{y}_\mathbf{x}(t), \boldsymbol{\alpha}(t)) = f(\mathbf{y}_\mathbf{x}(t), \boldsymbol{\alpha}(t))\, \boldsymbol{\alpha}(t)\,,$$

[3] It is assumed that $\overline{R}$ is strictly positive, g is positive, and are both Lipschitz-continuous functions.

where $f > 0$ is the speed of the entity and $\|\boldsymbol{\alpha}\| = 1$. The set of admissible controls is then

$$\mathcal{A} = \left\{\boldsymbol{\alpha} : \mathbb{R}^+ \cup \{0\} \mapsto S_1 \mid \boldsymbol{\alpha} \text{ is measurable}\right\},$$

where $\mathbb{R}^+$ is the set of positive real numbers, and $S_1 = \{\boldsymbol{\alpha} \in \mathbb{R}^n \mid \|\boldsymbol{\alpha}\| = 1\}$.

1.3 Parametrisation

For the remainder of this report, the path is considered to be parametrised by arc length s, which is more mathematically convenient than parametrisation with respect to time. When parametrised by s, the state equation simplifies to

$$\begin{aligned} \dot{\mathbf{y}}_{\mathbf{x}}(s) &= \boldsymbol{\alpha}(s) \quad (s > 0), \\ \mathbf{y}_{\mathbf{x}}(0) &= \mathbf{x}. \end{aligned} \tag{1}$$

The DPP becomes

$$u(\mathbf{x}) = \inf_{\boldsymbol{\alpha}\in\mathcal{A}} \left\{ \int_0^\tau R(\mathbf{y}_{\mathbf{x}}(s), \boldsymbol{\alpha}(s))\, ds + u(\mathbf{y}_{\mathbf{x}}(\tau)) \right\} \quad (0 < \tau \leqslant \ell_{\mathbf{x}}(\boldsymbol{\alpha})), \tag{DPP}$$

where $R(\mathbf{y}_{\mathbf{x}}(s), \boldsymbol{\alpha}(s)) = \overline{R}(\mathbf{y}_{\mathbf{x}}(s), \boldsymbol{\alpha}(s))\,/f(\mathbf{y}_{\mathbf{x}}(s), \boldsymbol{\alpha}(s))$ is the running cost per unit length, and $\ell_{\mathbf{x}}(\boldsymbol{\alpha})$ is the arc length associated with $\boldsymbol{\alpha}$ and $\mathbf{x}$ (shortest total arc length for $\mathbf{y}_{\mathbf{x}}(s, \boldsymbol{\alpha})$ to hit the target set). Finally, the HJB equation simplifies to

$$\begin{cases} \min\limits_{\boldsymbol{\alpha}\in\mathcal{A}} \{\boldsymbol{\alpha} \cdot \nabla u(\mathbf{x}) + R(\mathbf{x}, \boldsymbol{\alpha})\} = 0 & \text{in } \Omega \setminus \mathcal{T}, \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \mathcal{T}. \end{cases} \tag{HJB}$$

1.4 Isotropic Running Costs

If $R(\mathbf{x}, \boldsymbol{\alpha}) = R(\mathbf{x})$ then the running cost is isotropic, and the optimal control $\boldsymbol{\alpha}^*$ is the direction of steepest descent of u, given by

$$\boldsymbol{\alpha}^*(\mathbf{x}) = -\frac{\nabla u(\mathbf{x})}{\|\nabla u(\mathbf{x})\|}. \tag{2}$$

Equivalently, $\boldsymbol{\alpha}^*(\mathbf{x})$ is orthogonal to the level sets (iso-cost contours) of $u(\mathbf{x})$, if $u(\mathbf{x})$ is differentiable at $\mathbf{x}$. Equation (2) decouples the tasks of computing the optimal control and the value function, and results in the HJB equation simplifying to

$$\|\nabla u(\mathbf{x})\| = R(\mathbf{x}). \tag{3}$$

Equation (3) is known as the Eikonal equation and has numerous applications, for example, in path planning, computational geometry, computer vision, and image enhancement [Sethian 1999*b*].

Numerical methods for solving the Eikonal equation include Tsitsiklis' control-theoretic algorithm [Tsitsiklis 1995], Fast Marching Methods [Sethian 1999*a*, Sethian 1999*b*, Cristiani 2009], and Fast Sweeping Methods [Tsai et al. 2003, Kao, Osher & Tsai 2005, Qian, Zhang & Zhao 2007*a*, Qian, Zhang & Zhao 2007*b*]. While these numerical methods can be applied to anisotropic path planning problems for some special cases, their application to general anisotropic path planning problems is either not valid or impractical [Sethian & Vladimirsky 2003, Alton & Mitchell 2012].

2 Ordered Upwind Method

OUM compute the numerical value function U by processing subsets of the computational mesh. These include three disjoint sets: where U is known; where a tentative value for U has been computed using an approximation to the DPP; and where no information regarding U is known. The OUM algorithm iteratively updates these (and other) sets until U has been computed for all points in the computational mesh.

This section introduces OUM by presenting a summary of Section 6 from Sethian & Vladimirsky [2003], where minimum time optimal control problems are considered.[4]

2.1 Computational Mesh

To simplify the discussion let $\Omega \subset \mathbb{R}^2$, noting that OUM are valid in $\mathbb{R}^n$. Let $X_h \subset \mathbb{R}^2$ be an unstructured triangulated mesh of diameter h, where h is the maximum distance between any two adjacent mesh points, then adjacent mesh points $\mathbf{x}_j$ and $\mathbf{x}_k$ in X_h satisfy $\|\mathbf{x}_j - \mathbf{x}_k\| \leqslant h$. Define $\Omega_h = \Omega \cap X_h$ and $\mathcal{T}_h = \mathcal{T} \cap X_h$ to be discretisations of Ω and $\mathcal{T}$, respectively. Since $\mathcal{T} \subset \Omega$ the computational mesh is given by Ω_h.

2.2 Upwinding Approximation to the DPP

Let $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$ be a simplex with $\mathbf{x}_j, \mathbf{x}, \mathbf{x}_k \in \Omega_h$, such that $\mathbf{x}_j$, $\mathbf{x}_k$ are adjacent and $U(\mathbf{x}_j)$, $U(\mathbf{x}_k)$ are known from previous iterations of the OUM. At the heart of OUM is the following first order upwind approximation of the DPP in $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$:

$$V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x}) = \min_{\zeta\in[0,1]} \{\tau(\zeta)R(\mathbf{x}, \boldsymbol{\alpha}_\zeta) + \zeta U(\mathbf{x}_j) + (1-\zeta)U(\mathbf{x}_k)\}, \tag{4}$$

where $V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$ is the tentative value function in $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$, $\tau(\zeta) = \|\zeta\mathbf{x}_j + (1-\zeta)\mathbf{x}_k - \mathbf{x}\|$, and the control is given by

$$\boldsymbol{\alpha}_\zeta = \frac{\zeta\mathbf{x}_j + (1-\zeta)\mathbf{x}_k - \mathbf{x}}{\tau(\zeta)}.$$

Equation (4) can be motivated by observing that as $h \to 0$, the optimal path at $\mathbf{x}$ and within $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$ approaches a straight line that will intersect the line segment $\mathbf{x}_j\mathbf{x}_k$ at a point $\tilde{\mathbf{x}} = \zeta\mathbf{x}_j + (1-\zeta)\mathbf{x}_k$ for $\zeta \in [0,1]$. The value function at $\tilde{\mathbf{x}}$ is approximated using linear interpolation:

$$u(\tilde{\mathbf{x}}) \approx \zeta u(\mathbf{x}_j) + (1-\zeta)u(\mathbf{x}_k).$$

Furthermore, as $h \to 0$ the running cost approaches a constant in $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$. Applying these approximations to the DPP leads to Equation (4).

The tentative value function $V(\mathbf{x})$ is defined to be the minimum of $V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$ over a set of simplices obtained by varying $\mathbf{x}_j$ and $\mathbf{x}_k$; both $V(\mathbf{x})$ and the set of simplices are updated at each iteration of the OUM. When $V(\mathbf{x})$ becomes the minimum tentative value function during the iterations of the OUM, then the final value of $U(\mathbf{x})$ is identified with $V(\mathbf{x})$.

[4] The equivalence of minimum cost and minimum time optimal control problems is established in Vladimirsky [2001, Section 2.2.5].

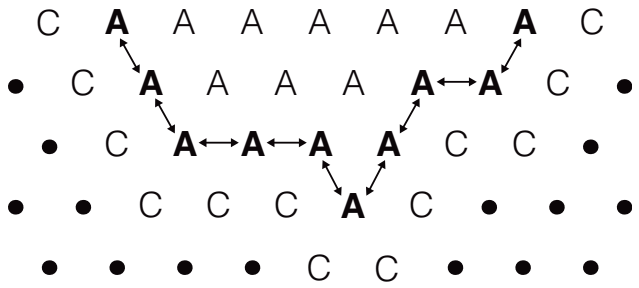


Figure 1: *An illustration of the Accepted (A), AcceptedFront (**A**), Considered (C) and Far (dots) sets. The line segments of the AF set are indicated by lines with arrowheads.*

2.3 OUM Sets

Like Dijkstra's shortest-path algorithm for graphs, the mesh points in Ω_h are partitioned into three sets that are updated at each iteration of the OUM:

Far: no information regarding U is known;

Accepted: U has been computed; and

Considered: adjacent to *Accepted* and tentative values, V, for U have been computed.

OUM also update the following sets at each iteration:

AcceptedFront: *Accepted* mesh points that are adjacent to some *Considered* points; and

AF: set of line segments $\mathbf{x}_j\mathbf{x}_k$, where $\mathbf{x}_j$ and $\mathbf{x}_k$ are adjacent mesh points in the *AcceptedFront* such that there exists a *Considered* point adjacent to both $\mathbf{x}_j$ and $\mathbf{x}_k$.

The *AF* set may be viewed as the set of line segments that constitute the boundary of the accepted region. An illustration of the *Accepted*, *AcceptedFront*, *AF*, *Considered* and *Far* sets is shown in Figure 1.

Finally, the near front set $\mathrm{NF}(\mathbf{x})$ is updated for each $\mathbf{x} \in$ *Considered* at each iteration of the OUM, and is defined to be

$$\mathrm{NF}(\mathbf{x}) = \{\mathbf{x}_j\mathbf{x}_k \in AF \mid \text{there exists a } \tilde{\mathbf{x}} \text{ on } \mathbf{x}_j\mathbf{x}_k \text{ such that } \|\tilde{\mathbf{x}} - \mathbf{x}\| \leqslant h\Upsilon(\mathbf{x})\}, \tag{5}$$

where $\Upsilon(\mathbf{x})$ is known as the anisotropy function, which is a local measure of anisotropy in the running cost:[5]

$$\Upsilon(\mathbf{x}) = \frac{\max_{\boldsymbol{\alpha}\in\mathcal{A}} R(\mathbf{x}, \boldsymbol{\alpha})}{\min_{\boldsymbol{\alpha}\in\mathcal{A}} R(\mathbf{x}, \boldsymbol{\alpha})}. \tag{6}$$

The $\mathrm{NF}(\mathbf{x})$ set is the part of AF that is relevant to $\mathbf{x}$, that is, $\mathrm{NF}(\mathbf{x})$ contains those line segments from which simplices are constructed with each $\mathbf{x} \in$ *Considered*, at which the tentative value function $V(\mathbf{x})$ is updated using Equation (4) at each iteration of the OUM.

[5]For isotropic running costs $\Upsilon(\mathbf{x}) = 1$.

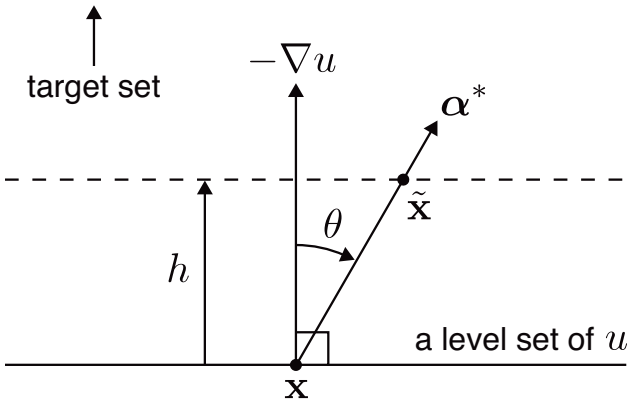


Figure 2: *Definition diagram for illuminating the origins of the* NF($\mathbf{x}$) *set.*

The remainder of this section is dedicated to illuminating the origins of the NF($\mathbf{x}$) set: NF($\mathbf{x}$) is crucial to the efficient computation of the numerical value function for anisotropic path planning problems.

Recall that for isotropic path planning problems, the optimal control $\boldsymbol{\alpha}^*$ and $-\nabla u$ are parallel (see Section 1.4), which is exploited to optimise the computational performance of numerical methods for the isotropic case. Whereas in the anisotropic case, $\boldsymbol{\alpha}^*$ and $-\nabla u$ are not necessarily parallel. This is the principal challenge for designing efficient numerical methods for anisotropic path planning problems: unlike in the isotropic case, to compute an optimal path at $\mathbf{x}$, it may be necessary to use mesh points that are not neighbours of $\mathbf{x}$.

A significant contribution of Sethian & Vladimirsky [2003] is the determination of a bound on the angle between $\boldsymbol{\alpha}^*$ and $-\nabla u$. This bound motivates the definition of the NF($\mathbf{x}$) set, and facilitates the efficient computation of the numerical value function by reducing the size of the computational stencil at $\mathbf{x}$.[6]

To derive an expression for the angle between $\boldsymbol{\alpha}^*$ and $-\nabla u$, consider a point $\mathbf{x}$ on a level set of u where ∇u exists, with $\mathbf{x}$ at a distance greater than $h > 0$ from the target set. Imagine a line drawn parallel to, and at a distance of h from, the tangent line to the level set at $\mathbf{x}$. If h is sufficiently small, then the optimal path at $\mathbf{x}$ can be approximated by a straight line that passes through $\tilde{\mathbf{x}}$, as shown in Figure 2. The angle between $\boldsymbol{\alpha}^*$ and $-\nabla u$ satisfies

$$\cos\theta = \frac{h}{\|\tilde{\mathbf{x}} - \mathbf{x}\|}, \tag{7}$$

as indicated in Figure 2.

It remains to eliminate θ from Equation (7). Following Vladimirsky [2001] and Sethian & Vladimirsky [2003], let $R_1(\mathbf{x}) = \min_{\boldsymbol{\alpha}\in\mathcal{A}} R(\mathbf{x}, \boldsymbol{\alpha})$ and $R_2(\mathbf{x}) = \max_{\boldsymbol{\alpha}\in\mathcal{A}} R(\mathbf{x}, \boldsymbol{\alpha})$, and recall that $R(\mathbf{x}, \boldsymbol{\alpha}) > 0$. It follows from the HJB equation that $\boldsymbol{\alpha}^* \cdot \nabla u < 0$ and consequently

$$\frac{\boldsymbol{\alpha}^* \cdot \nabla u}{R(\mathbf{x}, \boldsymbol{\alpha}^*)} \geqslant \frac{\boldsymbol{\alpha}^* \cdot \nabla u}{R_1(\mathbf{x})}. \tag{8}$$

[6]The computational stencil at $\mathbf{x}$ is the set of mesh points required to compute the numerical solution at $\mathbf{x}$. The NF($\mathbf{x}$) set defines the computational stencil at $\mathbf{x}$ for OUM.

Also, for all $\boldsymbol{\alpha} \in \mathcal{A}$,

$$\frac{\boldsymbol{\alpha}^* \cdot \nabla u}{R(\mathbf{x}, \boldsymbol{\alpha}^*)} \leqslant \frac{\boldsymbol{\alpha} \cdot \nabla u}{R(\mathbf{x}, \boldsymbol{\alpha})}, \tag{9}$$

since u satisfies the HJB equation and $\boldsymbol{\alpha}^*$ is an optimal control. Choosing $\boldsymbol{\alpha} = -\nabla u / \|\nabla u\|$ in Equation (9) leads to

$$\frac{\boldsymbol{\alpha}^* \cdot \nabla u}{R(\mathbf{x}, \boldsymbol{\alpha}^*)} \leqslant -\frac{\|\nabla u\|}{R_2(\mathbf{x})}. \tag{10}$$

Combining Equations (8) and (10) results in

$$\boldsymbol{\alpha}^* \cdot (-\nabla u) \geqslant \frac{\|\nabla u\|}{\Upsilon(\mathbf{x})},$$

where $\Upsilon(\mathbf{x})$ is given by Equation (6). By definition, $\boldsymbol{\alpha}^* \cdot (-\nabla u) = \|\nabla u\| \cos\theta$ and therefore

$$\cos\theta \geqslant \frac{1}{\Upsilon(\mathbf{x})}. \tag{11}$$

Equation (11) is the bound on the angle between $\boldsymbol{\alpha}^*$ and $-\nabla u$ reported in Vladimirsky [2001] and Sethian & Vladimirsky [2003]. Finally, Equations (7) and (11) yield

$$\|\tilde{\mathbf{x}} - \mathbf{x}\| \leqslant h\Upsilon(\mathbf{x}). \tag{12}$$

Now let $\mathbf{x} \in \textit{Considered}$. Returning to the upwinding approximation to the DPP (refer to Section 2.2), it can be seen from Equation (4) and Figure 1 that for $V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$ to be well-defined, $\mathbf{x}_j\mathbf{x}_k \in AF$. Hence a candidate definition for the tentative value function $V(\mathbf{x})$ is

$$V(\mathbf{x}) = \min_{\mathbf{x}_j\mathbf{x}_k \in AF} V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x}). \tag{13}$$

Indeed, an algorithm can be developed to compute U based on Equation (13) such that $U \to u$ as $h \to 0$ [Sethian & Vladimirsky 2003]; however, this algorithm will not be computationally efficient. Instead, if $\tilde{\mathbf{x}} = \zeta\mathbf{x}_j + (1-\zeta)\mathbf{x}_k$ for $\zeta \in [0,1]$, then Equation (12) can be employed to reduce the number of elements in AF that are required to evaluate $V(\mathbf{x})$, that is,

$$V(\mathbf{x}) = \min_{\mathbf{x}_j\mathbf{x}_k \in \mathrm{NF}(\mathbf{x})} V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x}), \tag{14}$$

which follows from Equation (5). OUM use this definition of V to compute U.

Note that the aforementioned arguments only illuminate the definition of $\mathrm{NF}(\mathbf{x})$; in particular, these arguments fail at points where ∇u does not exist. The proof that Equations (13) and (14) result in the same value for U appears in Sethian & Vladimirsky [2003].

2.4 The OUM Algorithm

The implementation of the OUM algorithm that is the subject of this report appears in Section 3.4 as Algorithm 8. The OUM algorithm from Sethian & Vladimirsky [2003] is reproduced here to serve as an introduction and to enable a comparison with Algorithm 8:

1. Start with all the mesh points in *Far*.
2. Move the mesh points in the target set $\mathbf{x} \in \mathcal{T}_h$ to *Accepted* ($U(\mathbf{x}) = g(\mathbf{x})$).
3. Move all the mesh points $\mathbf{x}$ adjacent to the target set into *Considered* and evaluate the tentative values
$$V(\mathbf{x}) := \min_{\mathbf{x}_j\mathbf{x}_k \in \mathrm{NF}(\mathbf{x})} V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x}). \tag{15}$$
4. Find the mesh point $\bar{\mathbf{x}}$ with the smallest value of V among all the *Considered*.
5. Move $\bar{\mathbf{x}}$ to *Accepted* ($U(\bar{\mathbf{x}}) = V(\bar{\mathbf{x}})$) and update the *AcceptedFront*.
6. Move the *Far* mesh points adjacent to $\bar{\mathbf{x}}$ into *Considered* and compute their tentative values by Equation (15).
7. Recompute the value for all the other *Considered* $\mathbf{x}$ such that $\bar{\mathbf{x}}\mathbf{x}_i \in \mathrm{NF}(\mathbf{x})$
$$V(\mathbf{x}) := \min\left\{V(\mathbf{x}), \min_{\bar{\mathbf{x}}\mathbf{x}_i \in \mathrm{NF}(\mathbf{x})} V_{\bar{\mathbf{x}}\mathbf{x}_i}(\mathbf{x})\right\}. \tag{16}$$
8. If *Considered* is not empty, then go to 4.

The OUM algorithm

- has a computational complexity of $O(\Upsilon M \log(M))$ as $h \to 0$;
- computes the numerical solution U that converges to u as $h \to 0$;
- is at most first-order accurate; and
- is valid for anisotropic path planning problems.

Here M is the number of mesh points in Ω_h and $\Upsilon = \max_{\mathbf{x}\in\Omega} \Upsilon(\mathbf{x})$. This estimate of the computational complexity counts the calculation of $V_{\mathbf{x}_i\mathbf{x}_j}(\mathbf{x})$ via Equation (4) as a single operation, since this is performed independently of the other mesh points in Ω_h.

Note that Dijkstra's algorithm has a computational complexity of $O(M \log(M))$. However, unlike Dijkstra's algorithm, and Sethian's Fast Marching Method [Sethian 1999*a*, Sethian 1999*b*] and Tsitsiklis' algorithm [Tsitsiklis 1995], the OUM algorithm does *not* add mesh points to the *Accepted* set in the order of increasing U.

3 Implementation

This section details a generic implementation of the OUM that has the following characteristics:

- Ω_h is constructed from a uniform mesh of equilateral triangles on $\mathbb{R}^2$; and
- U and V are stored as functions of the integer mesh co-ordinates to allow consistent and efficient data retrieval.[7]

The requirements to use a non-uniform mesh are typically driven by features of a particular application, such as the geometry of the domain and the form of the running cost, or by the need to improve computational performance. Generating a non-uniform mesh requires a refinement condition, a refinement algorithm, and data structures for efficient storage and retrieval of geometric and topological properties of the mesh. For OUM, at least the position and adjacency of the mesh points must be stored.

A uniform mesh is considered in this report to simplify the discussion, avoid application specific issues, and to demonstrate a concrete implementation of the OUM. A uniform mesh of equilateral triangles was chosen because the distance between adjacent mesh points is equal in all directions. Whereas for a triangulated mesh constructed from a Cartesian grid with spacing δ, the triangulation diameter is increased to $\sqrt{2}\delta$ and hence the numerical accuracy is reduced. However, a triangulation based on a Cartesian grid is easier to implement and generalise to higher dimensions, and many of the standard approaches for visualising the optimal paths and level sets are based on Cartesian grids. Hence an implementation of the mesh-dependent functions for this case has been included in Appendix A.

For more information on mesh construction, including mesh refinement and construction in higher dimensions, refer to Cline & Renka [1984], Bänsch [1991], Bey [1995], Maubach [1995], Ruppert [1995], Arnold, Mukherjee & Pouly [2000], Shewchuk [2002] and Hjelle & Dæhlen [2006].

Define the set of integer mesh co-ordinates $\Omega_h^{\mathbb{Z}}$ to be

$$\Omega_h^{\mathbb{Z}} = \big\{(i,j) \in \mathbb{Z}^2 \mid \mathbf{x}(i,j) \in \Omega_h\big\},$$

where $\mathbf{x}(i,j)$ is a mesh point with integer mesh co-ordinate (i,j). Since U and V are stored as functions of $\Omega_h^{\mathbb{Z}}$, for the remainder of this report, the OUM sets introduced in Section 2.3 are considered to be subsets of $\Omega_h^{\mathbb{Z}}$ (or constructed from elements of $\Omega_h^{\mathbb{Z}}$), and hence the OUM algorithm will process these subsets to compute U.

3.1 Mesh Functions

Functions that depend on the structure of the triangulation are presented in this section. This comprises functions for transforming between mesh points $\mathbf{x}(i,j) \in \Omega_h$ and mesh co-

[7] For example, U and V could be stored in an array, with the integer mesh co-ordinates corresponding to indices of the array. Alternatively, some mathematical software packages allow functions to be defined for each discrete point in their domain; in which case, these functions effectively behave like arrays with arbitrary indices.

ordinates $(i,j) \in \Omega_h^{\mathbb{Z}}$, for mesh construction, and functions that return mesh co-ordinates corresponding to the neighbours of given mesh points.

3.1.1 Mesh Transformations

The mesh point $\mathbf{x}(i,j) \in \Omega_h$ indexed by the mesh co-ordinate $(i,j) \in \Omega_h^{\mathbb{Z}}$ is given by[8]

$$\mathbf{x}(i,j) = \mathbf{x}_{\mathcal{T}} + h\big(i\mathbf{e}_1^{\triangle} + j\mathbf{e}_2^{\triangle}\big), \tag{17}$$

where $\mathbf{x}_{\mathcal{T}} \in \mathrm{int}(\mathcal{T})$ is the target location and

$$\mathbf{e}_1^{\triangle} = (1,0), \qquad \mathbf{e}_2^{\triangle} = \frac{1}{2}(1,\sqrt{3}), \tag{18}$$

are the triangulation unit vectors. Observe that $\mathbf{e}_1^{\triangle} \cdot \mathbf{e}_2^{\triangle} = 1/2$.

The mesh co-ordinate $(i,j) \in \Omega_h^{\mathbb{Z}}$ which results in a mesh point $\mathbf{x}(i,j) \in \Omega_h$ that is nearest to a given point $\mathbf{y} \in \Omega$ (not necessarily a mesh point) is obtained from[9]

$$\begin{aligned} i &= \left[\frac{1}{h}(\mathbf{y} - \mathbf{x}_{\mathcal{T}}) \cdot \mathbf{e}_1 - \frac{1}{2}j\right], \\ j &= \left[\frac{2}{\sqrt{3}}\frac{1}{h}(\mathbf{y} - \mathbf{x}_{\mathcal{T}}) \cdot \mathbf{e}_2\right]. \end{aligned} \tag{19}$$

Here $[\,\cdot\,]$ represents rounding to the nearest integer, and $\mathbf{e}_i$ is the ith standard basis vector in $\mathbb{R}^2$.

Equations (17) to (19) enable transformations between subsets of $\Omega_h^{\mathbb{Z}}$ and subsets of Ω to be defined. Let `MeshPoint` take as inputs h, $\mathbf{x}_{\mathcal{T}}$ and mesh co-ordinates, and return the corresponding mesh points, and let `NearestIndex` take as inputs h, $\mathbf{x}_{\mathcal{T}}$ and points in Ω, and return the corresponding (nearest) mesh co-ordinates, that is:

$$\texttt{MeshPoint} : \mathbb{R}^{0,1} \times \mathrm{int}(\mathcal{T}) \times \Omega_h^{\mathbb{Z}} \longrightarrow \Omega_h,$$

$$\texttt{NearestIndex} : \mathbb{R}^{0,1} \times \mathrm{int}(\mathcal{T}) \times \Omega \longrightarrow \Omega_h^{\mathbb{Z}},$$

where $\mathbb{R}^{0,1} = \{x \in \mathbb{R} \mid 0 < x < 1\}$. In particular, if $\mathcal{S}_h^{\mathbb{Z}} \subseteq \Omega_h^{\mathbb{Z}}$ and

$$\texttt{MeshPoint}\Big(h, \mathbf{x}_{\mathcal{T}}, \mathcal{S}_h^{\mathbb{Z}}\Big) = \mathcal{S}_h,$$

where $\mathcal{S}_h \subseteq \Omega_h$, then

$$\texttt{NearestIndex}(h, \mathbf{x}_{\mathcal{T}}, \mathcal{S}_h) = \mathcal{S}_h^{\mathbb{Z}}.$$

However, if $\mathcal{S} \subseteq \Omega$, then in general

$$\texttt{MeshPoint}(h, \mathbf{x}_{\mathcal{T}}, \texttt{NearestIndex}(h, \mathbf{x}_{\mathcal{T}}, \mathcal{S})) \neq \mathcal{S},$$

as $\mathcal{S}$ may contain elements that are not mesh points.

[8] The dependence of $\mathbf{x}(i,j)$ on the target location and the triangulation diameter is suppressed for notational simplicity.

[9] It can be shown that $\|\mathbf{y} - \mathbf{x}(i,j)\|$ is minimised for all the combinations of rounding i and j up and down to the nearest integer, if (i,j) is given by Equation (19).

Algorithm 1: Construct $\Omega_h^{\mathbb{Z}}$ where Ω is a unit square centred at $\mathbf{x}_c$.

Input: $\rho_h(\Omega)$, h, $\mathbf{x}_{\mathcal{T}}$ and $\mathbf{x}_c$ such that $\|\mathbf{x}_{\mathcal{T}} - \mathbf{x}_c\|_\infty \leqslant 0.5$
Output: $\Omega_h^{\mathbb{Z}}$

```
Ω_h^ℤ ← ∅
foreach (i, j) ∈ ℤ² such that |i| ⩽ ρ_h(Ω) and |j| ⩽ ρ_h(Ω) do
    if ‖MeshPoint(h, x_T, (i, j)) − x_c‖_∞ ⩽ 0.5 then
        Ω_h^ℤ ← Ω_h^ℤ ∪ {(i, j)}
```

3.1.2 Mesh Construction

Mesh construction strongly depends on the geometry of Ω and $\mathcal{T}$, even for a uniform mesh, and a general discussion of this topic is beyond the scope of this report. Instead, a construction where Ω is a unit square is presented here as an example.

Recall that the OUM algorithm processes subsets of $\Omega_h^{\mathbb{Z}}$ to compute U. To construct $\Omega_h^{\mathbb{Z}}$ it is necessary to determine the magnitudes of i and j such that X_h covers Ω. To this end, let r be the largest distance between any two elements of Ω. Setting $\mathbf{y} - \mathbf{x}_{\mathcal{T}} = r(\cos\theta, \sin\theta)$ in Equation (19) leads to

$$|i| \leqslant \left\lceil \frac{2}{\sqrt{3}} \frac{r}{h} \right\rceil \text{ and } |j| \leqslant \left\lceil \frac{2}{\sqrt{3}} \frac{r}{h} \right\rceil,$$

where $\lceil \cdot \rceil$ represents rounding up to the nearest integer. Therefore, define the computational radius $\rho_h(\Omega)$ to be

$$\rho_h(\Omega) = \left\lceil \frac{2}{\sqrt{3}} \frac{r(\Omega)}{h} \right\rceil.$$

A construction where Ω is a unit square centred at $\mathbf{x}_c$ is presented in Algorithm 1; in this case $r(\Omega) = \sqrt{2}$. Observe that Algorithm 1 can be used to construct $\Omega_h^{\mathbb{Z}}$ when Ω is a circle of radius $1/2$ centred at $\mathbf{x}_c$ by setting $r(\Omega) = 1$ and replacing $\|\cdot\|_\infty$ with $\|\cdot\|$.

3.1.3 Neighbourhood Functions

The notion of adjacency is central to the OUM algorithm, and therefore a function that returns the neighbours of a set is required. To begin, the neighbours of the mesh coordinate (i, j) are given by[10]

$$N(i,j) = \{(i+1,j), (i,j+1), (i-1,j+1), (i-1,j), (i,j-1), (i+1,j-1)\}, \tag{20}$$

noting that $(i, j) \notin N(i, j)$. The $N(i, j)$ set is shown in Figure 3 mapped onto Ω_h. Observe that the elements of $N(i, j)$ are arranged anticlockwise when mapped onto Ω_h. The neighbourhood of (i, j) is given by

$$\overline{N}(i,j) = \{(i,j), (i+1,j), (i,j+1), (i-1,j+1), (i-1,j), (i,j-1), (i+1,j-1)\},$$

[10] If (i, j) is on the boundary of $\Omega_h^{\mathbb{Z}}$ then some elements of $N(i, j)$ will not belong to $\Omega_h^{\mathbb{Z}}$. This does not cause any difficulties since $N(i, j)$ is always intersected with a subset of $\Omega_h^{\mathbb{Z}}$ in this report.

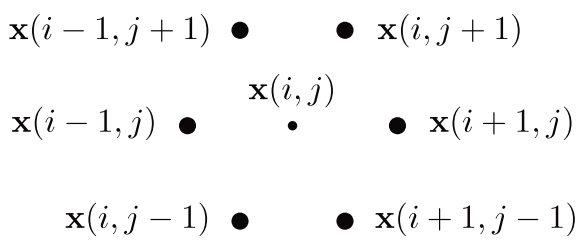


Figure 3: *The $N(i,j)$ set shown mapped onto Ω_h as large dots, where the $\mathbf{x}(\,\cdot\,,\,\cdot\,)$ are given by Equation (17).*

that is, $\overline{N}(i,j) = N(i,j) \cup \{(i,j)\}$.

The definition of $N(i,j)$ can be generalised to encompass sets by defining $N(\mathcal{S}_h^{\mathbb{Z}})$ to be

$$N(\mathcal{S}_h^{\mathbb{Z}}) = \bigcup_{(i,j)\in\mathcal{S}_h^{\mathbb{Z}}} N(i,j) \setminus \mathcal{S}_h^{\mathbb{Z}}, \tag{21}$$

for $\mathcal{S}_h^{\mathbb{Z}} \subset \Omega_h^{\mathbb{Z}}$. In this report, $N(\mathcal{S}_h^{\mathbb{Z}})$ is constructed using

$$N(\mathcal{S}_h^{\mathbb{Z}}) = \bigcup_{(i,j)\in\mathcal{S}_h^{\mathbb{Z}}} N(i,j).$$

This results in $N(\mathcal{S}_h^{\mathbb{Z}})$ containing elements of $\mathcal{S}_h^{\mathbb{Z}}$, which is inconsistent with the definition provided by Equation (21). However this does not cause any problems, because if $\mathcal{S}_h^{\mathbb{Z}}$ is a singleton set then the correct result is obtained, otherwise $N(\mathcal{S}_h^{\mathbb{Z}})$ will always be intersected with a set that is disjoint to $\mathcal{S}_h^{\mathbb{Z}}$ in this report.

3.2 Set Construction

This section is devoted to the construction of the OUM sets, which were introduced in Section 2.3.

3.2.1 Adjacency Function

To reiterate, the notion of adjacency is central to the OUM algorithm. A mesh co-ordinate (i,j) is defined to be adjacent to a set $B_h^{\mathbb{Z}} \subset \Omega_h^{\mathbb{Z}}$ if $(i,j) \in N(B_h^{\mathbb{Z}})$, where $N(\,\cdot\,)$ is given by Equation (21). The adjacency function $\texttt{AdjacentTo}(A_h^{\mathbb{Z}}, B_h^{\mathbb{Z}})$ is then defined to be the set of mesh co-ordinates in $A_h^{\mathbb{Z}} \subset \Omega_h^{\mathbb{Z}}$ that are adjacent to the mesh co-ordinates in $B_h^{\mathbb{Z}}$:

$$\texttt{AdjacentTo}(A_h^{\mathbb{Z}}, B_h^{\mathbb{Z}}) = A_h^{\mathbb{Z}} \cap N(B_h^{\mathbb{Z}}). \tag{22}$$

The `AdjacentTo` function is required to construct most of the OUM sets.

Algorithm 2: `AFAdjacentPairs`

Input: *AcceptedFront* and *Considered*
Output: *AF*

```
1  AF ← ∅
2  if Considered ≠ ∅ then
3      S ← AcceptedFront
4      while S ≠ ∅ do
5          select (i, j) ∈ S
6          S ← S \ {(i, j)}
7          A ← AdjacentTo(S, {(i, j)})
8          N_ij ← Considered ∩ N(i, j)
9          foreach (k, l) ∈ A do
10             if N_ij ∩ N(k, l) ≠ ∅ then
11                 AF ← AF ∪ {{(i, j), (k, l)}}
```

3.2.2 *AcceptedFront* Construction

By definition, the *AcceptedFront* can be constructed using

$$\mathit{AcceptedFront} = \texttt{AdjacentTo}(\mathit{Accepted}, \mathit{Considered}). \tag{23}$$

However this is computationally inefficient, since changes to the *AcceptedFront* between iterations of the OUM algorithm only occur within a neighbourhood of the most recently accepted mesh co-ordinate $(\bar{i}, \bar{j})$. Consequently, after the *AcceptedFront* has been initialised on the mesh co-ordinates in the target set, it is then updated according to

$$\begin{aligned}\mathit{AcceptedFront} \leftarrow \big(\mathit{AcceptedFront} \setminus \overline{N}(\bar{i}, \bar{j})\big) \cup \\ \texttt{AdjacentTo}(\mathit{Accepted} \cap \overline{N}(\bar{i}, \bar{j}), \mathit{Considered}).\end{aligned} \tag{24}$$

Equation (24) is verified in Appendix B.

The update rule given by Equation (24) adds new elements to the *AcceptedFront* by calling `AdjacentTo` to operate on $\mathit{Accepted} \cap \overline{N}(\bar{i}, \bar{j})$, which has (at most) seven elements. Compare this with Equation (23), where the *AcceptedFront* is constructed from the entire *Accepted* set, which may have tens of thousands of elements.[11]

3.2.3 *AF* Construction

Recall from Section 2.3 that the AF set is given, in terms of mesh co-ordinates, by

$$\begin{aligned}AF = \big\{ \{(i,j),(k,l)\} \in \mathit{AcceptedFront} \times \mathit{AcceptedFront} \mid \\ \|\mathbf{x}(i,j) - \mathbf{x}(k,l)\| = h \text{ and } \mathit{Considered} \cap N(i,j) \cap N(k,l) \neq \emptyset \big\}.\end{aligned}$$

[11] The *Accepted* set grows by one element per iteration of the OUM algorithm until $\mathit{Accepted} = \Omega_h^{\mathbb{Z}}$.

Algorithm 3: `AFUpdate`

Input: AF, *AcceptedFront*, *Considered*, and $\overline{N}(\bar{i},\bar{j})$, where $(\bar{i},\bar{j})$ is the most recently accepted mesh co-ordinate
Output: updated AF

1 $P(\bar{i},\bar{j}) \leftarrow \emptyset$
2 **foreach** $(i,j) \in \overline{N}(\bar{i},\bar{j})$ **do**
3 $\quad A \leftarrow \texttt{AdjacentTo}(\overline{N}(\bar{i},\bar{j}), \{(i,j)\})$
4 $\quad$ **foreach** $(k,l) \in A$ **do**
5 $\quad\quad P(\bar{i},\bar{j}) \leftarrow P(\bar{i},\bar{j}) \cup \{\{(i,j),(k,l)\}\}$
6 $AF \leftarrow (AF \setminus P(\bar{i},\bar{j})) \cup \texttt{AFAdjacentPairs}(\mathit{AcceptedFront} \cap \overline{N}(\bar{i},\bar{j}), \mathit{Considered})$

Two functions are employed to construct AF: `AFAdjacentPairs` constructs the entire AF, and `AFUpdate` updates AF at step 5 of the OUM algorithm; refer to Section 2.4.

`AFAdjacentPairs` takes the *AcceptedFront* and *Considered* sets as inputs and returns the entire AF, and is defined in Algorithm 2. `AFAdjacentPairs` produces the AF set with minimal cardinality, meaning that if $\{(i,j),(k,l)\} \in AF$ then $\{(k,l),(i,j)\} \notin AF$, which is permitted since $\mathbf{x}_i\mathbf{x}_j$ is equivalent to $\mathbf{x}_j\mathbf{x}_i$; this is achieved at line 6 of Algorithm 2. `AFAdjacentPairs` can be exclusively used to construct AF. However, the computational performance of AF construction can be dramatically improved by noting that elements are only added to (or removed from) the *AcceptedFront* from a neighbourhood of the most recently accepted mesh co-ordinate $(\bar{i},\bar{j})$.

This is exploited in the definition of `AFUpdate`, which updates AF at step 5 of the OUM algorithm, and is presented in Algorithm 3. The purpose of lines 2–5 of Algorithm 3 is to construct the set $P(\bar{i},\bar{j})$, which consists of all adjacent pairs of mesh co-ordinates with elements in $\overline{N}(\bar{i},\bar{j})$, including elements that no longer belong to AF. $P(\bar{i},\bar{j})$ is given by[12]

$$P(\bar{i},\bar{j}) = \big\{\{(i,j),(k,l)\} \in \overline{N}(\bar{i},\bar{j}) \times \overline{N}(\bar{i},\bar{j}) \mid \|\mathbf{x}(i,j) - \mathbf{x}(k,l)\| = h\big\}.$$

`AFUpdate` adds new elements to AF by calling `AFAdjacentPairs` to operate on the set $\mathit{AcceptedFront} \cap \overline{N}(\bar{i},\bar{j})$, which has (at most) seven elements, whereas *AcceptedFront* may have thousands of elements at each iteration of the OUM algorithm. The validity of `AFUpdate` can be established by comparing line 6 of Algorithm 3 to the *AcceptedFront* update rule given by Equation (24).

3.2.4 $\mathrm{NF}(i,j)$ Construction

At step 7 of the OUM algorithm in Section 2.4, the tentative value function is recomputed for each $(i,j) \in \mathit{Considered}$ such that $\{(\bar{i},\bar{j}), \cdot\} \in \mathrm{NF}(i,j)$,[13] where $(\bar{i},\bar{j}) \in \mathit{Accepted}$ is the most recently accepted mesh co-ordinate. Recall from Equation (5) that, in terms of

[12] Note that if $\{(i,j),(k,l)\} \in P(\bar{i},\bar{j})$ then $\{(k,l),(i,j)\} \in P(\bar{i},\bar{j})$.
[13] For greater clarity, this should read $\{(\bar{i},\bar{j}), \cdot\} \in \mathrm{NF}(i,j)$ or $\{\cdot, (\bar{i},\bar{j})\} \in \mathrm{NF}(i,j)$.

Algorithm 4: `AFSubset`

Input: AF, and $(\bar{i}, \bar{j})$ corresponding to $\bar{\mathbf{x}}$ in Section 2.4
Output: subset of AF containing $\{\,\cdot\,, (\bar{i}, \bar{j})\}$ or $\{(\bar{i}, \bar{j}),\, \cdot\,\}$

1 $S \leftarrow \emptyset$
2 $N \leftarrow N(\bar{i}, \bar{j})$
3 **foreach** $(k, l) \in N$ **do**
4 $\quad S \leftarrow S \cup \{\{(k, l), (\bar{i}, \bar{j})\}, \{(\bar{i}, \bar{j}), (k, l)\}\}$
5 $AF \cap S$

mesh co-ordinates, the near front $\mathrm{NF}(i, j)$ is defined to be

$$\mathrm{NF}(i,j) = \big\{ \{(k,l),(m,n)\} \in AF \mid \text{there exists a } \tilde{\mathbf{x}} \text{ on } \mathbf{x}(k,l)\,\mathbf{x}(m,n) \text{ such that } \|\tilde{\mathbf{x}} - \mathbf{x}(i,j)\| \leqslant h\Upsilon(\mathbf{x}(i,j)) \big\}. \tag{25}$$

The near front is constructed once per element of the *Considered* set, which may contain thousands of elements, for each iteration of the OUM algorithm. Therefore improving the efficiency of near front construction will greatly improve the overall computational performance of the OUM algorithm. To this end, at step 7, instead of constructing $\mathrm{NF}(i, j)$ from the entire AF and then extracting the subset of $\mathrm{NF}(i, j)$ such that $\{(\bar{i}, \bar{j}),\, \cdot\,\} \in \mathrm{NF}(i, j)$, it is more efficient to construct $\mathrm{NF}(i, j)$ from the subset of AF that contains $(\bar{i}, \bar{j})$ in one of its mesh co-ordinate pairs. The `AFSubset` function takes as inputs AF and $(\bar{i}, \bar{j})$, and returns the subset of AF containing $\{\,\cdot\,, (\bar{i}, \bar{j})\}$ or $\{(\bar{i}, \bar{j}),\, \cdot\,\}$; see Algorithm 4. Although AF will only contain $\{\,\cdot\,, (\bar{i}, \bar{j})\}$ or $\{(\bar{i}, \bar{j}),\, \cdot\,\}$, it is not possible to know which one in advance; this is taken into account on line 4 of Algorithm 4.

Now all that remains is to derive a function to apply the conditions in Equation (25) to determine which elements of AF also belong to $\mathrm{NF}(i, j)$. To begin, let the quadratic polynomial $p(\zeta)$ be defined by

$$p(\zeta) = \|\zeta\mathbf{x}(k,l) + (1-\zeta)\mathbf{x}(m,n) - \mathbf{x}(i,j)\|^2 - (h\Upsilon(\mathbf{x}(i,j)))^2, \tag{26}$$

for $(i, j) \in \mathit{Considered}$ and $\{(k, l), (m, n)\} \in AF$, where the line segment $\mathbf{x}(k, l)\,\mathbf{x}(m, n)$ is given by $\zeta\mathbf{x}(k, l) + (1 - \zeta)\mathbf{x}(m, n)$ for $\zeta \in [0, 1]$. Therefore, if there exists a $\zeta \in [0, 1]$ such that

$$p(\zeta) \leqslant 0, \tag{27}$$

then $\{(k, l), (m, n)\} \in \mathrm{NF}(i, j)$.

Let $\Delta(p)$ be the discriminant of $p(\zeta)$. The following statements are properties of $p(\zeta)$:

- $p'' = 2h^2 > 0$, and hence $p(\zeta)$ is a convex function.
- If $\Delta(p) < 0$, then $p(\zeta) > 0$ for all ζ, and hence Equation (27) is not satisfied.
- If $\Delta(p) \geqslant 0$, then $p(\operatorname{argmin}(p(\zeta))) \leqslant 0$. Therefore if $\operatorname{argmin}(p(\zeta)) \in [0, 1]$, then Equation (27) is satisfied.

Algorithm 5: `NearFrontTest`

Input: h, $\{(k,l),(m,n)\} \in AF$, $(i,j) \in \mathit{Considered}$, and $\Upsilon(\mathbf{x}(i,j))$
Output: True if $\{(k,l),(m,n)\} \in \mathrm{NF}(i,j)$, otherwise False

$C_1 \leftarrow m - i$
$C_2 \leftarrow n - j$
$C_3 \leftarrow k - m$
$C_4 \leftarrow l - n$
$\mathrm{argmin}(p(\zeta)) \leftarrow -\frac{1}{2}(2C_1C_3 + C_1C_4 + C_2C_3 + 2C_2C_4)$
$p'' \leftarrow 2h^2$
$p'(0) \leftarrow -p''\mathrm{argmin}(p(\zeta))$
$p(0) \leftarrow \frac{1}{8}\Big(3p''C_2^2 + p''\left(2C_1 + C_2\right)^2 - 8\left(h\Upsilon(\mathbf{x}(i,j))\right)^2\Big)$
$p(1) \leftarrow \frac{1}{2}p'' + p'(0) + p(0)$
$\Delta(p) \leftarrow \left(p'(0)\right)^2 - 2p''p(0)$
if $\Delta(p) < 0$ **then**
  return False
else if $p(0) \leqslant 0$ or $p(1) \leqslant 0$ **then**
  return True
else if $0 \leqslant \mathrm{argmin}(p(\zeta)) \leqslant 1$ **then**
  return True
else
  return False

- If $\Delta(p) \geqslant 0$, $\mathrm{argmin}(p(\zeta)) < 0$ and $p(0) > 0$, then $p(\zeta) > 0$ for $\zeta \in [0,1]$ and Equation (27) is not satisfied.
- If $\Delta(p) \geqslant 0$, $\mathrm{argmin}(p(\zeta)) > 1$ and $p(1) > 0$, then $p(\zeta) > 0$ for $\zeta \in [0,1]$ and Equation (27) is not satisfied.

Also, since $p(\zeta)$ is a quadratic polynomial,

$$\Delta(p) = \big(p'(0)\big)^2 - 2p''p(0),$$

$$\mathrm{argmin}(p(\zeta)) = -\frac{p'(0)}{p''},$$

$$p(1) = \frac{1}{2}p'' + p'(0) + p(0).$$

Finally, using Equations (17) and (18) and the definition of $p(\zeta)$ leads to

$$p(0) = h^2\big((m-i)^2 + (n-j)^2 + (m-i)(n-j)\big) - \left(h\Upsilon(\mathbf{x}(i,j))\right)^2,$$

$$p'(0) = 2h^2\bigg((m-i)(k-m) + \frac{1}{2}(m-i)(l-n) + \frac{1}{2}(n-j)(k-m) + (n-j)(l-n)\bigg).$$

Algorithm 6: `NearFront`

Input: h, $\mathbf{x}_{\mathcal{T}}$, AF, and $(i,j) \in \mathit{Considered}$
Output: $\mathrm{NF}(i,j)$

```
1 NF(i,j) ← ∅
2 C ← ϒ(MeshPoint(h, x_T, (i,j)))
3 foreach {(k,l),(m,n)} ∈ AF do
4     if NearFrontTest(h, {(k,l),(m,n)}, (i,j), C) then
5         NF(i,j) ← NF(i,j) ∪ {{(k,l),(m,n)}}
```

These properties of $p(\zeta)$ are incorporated into the definition of `NearFrontTest`, which returns "True" if an element of AF is also an element of $\mathrm{NF}(i,j)$, or otherwise returns "False"; see Algorithm 5.[14] `NearFrontTest` is a major contributor to the total run-time of the OUM algorithm, due to the number of calls to `NearFrontTest`. Therefore even small improvements to the efficiency of `NearFrontTest` may result in large improvements to the computational performance of the OUM algorithm.

$\mathrm{NF}(i,j)$ can now be constructed by calling `NearFrontTest`. This is performed by the `NearFront` function, which is defined in Algorithm 6.

3.3 Tentative Value Function

This section concerns the evaluation of the tentative value function V in Equation (15) of the OUM algorithm presented in Section 2.4. To evaluate the tentative value function, two minimisations are performed: locally within each simplex constructed from the near front, and then the minimum of these values is selected.

3.3.1 Local Minimisation

To evaluate V, it is first necessary to determine $V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$ by performing the function minimisation in Equation (4). This local minimisation is performed using an algorithm that emulates the Golden Section Search routine from Press et al. [1992].

The Golden Section Search algorithm's inputs include a numerical precision and an initial bracket for the minimising abscissa. The numerical precision of this algorithm is set to h in our implementation. This incurs an $O(h^2)$ numerical error in $V_{\mathbf{x}_j\mathbf{x}_k}(\mathbf{x})$, which is acceptable as OUM are at most first-order accurate.[15]

The routine for initially bracketing a minimum presented in Press et al. [1992] is for functions defined on an unbounded interval, which is not the case here. This bracketing routine has been modified to restrict the domain to $[0,1]$ in our implementation of

[14] As $\mathbf{x}(i,j)$ and $\mathbf{x}(m,n)$ are not necessarily adjacent (see Section 2.2), C_1 and C_2 in Algorithm 5 may be large and therefore it is desirable to avoid squaring these terms. Consequently, on line 8 of Algorithm 5, $p(0)$ is expressed in a form to enable cancellation between C_1 and C_2 to occur before these terms are squared.

[15] This follows from the uniform Lipschitz continuity of U [Sethian & Vladimirsky 2003].

the OUM.[16] As in Press et al. [1992], the resulting bracketing algorithm is somewhat "formidable", and hence it is not included in this report. Furthermore, experience suggests that the results produced by the OUM algorithm when using this bracketing algorithm compared with simply providing the Golden Section Search algorithm with an initial bracket of $(0, 0.5, 1)$, are numerically indistinguishable for the examples considered in this report. This is not surprising because as $h \to 0$, the function in the braces in Equation (4) will approach a straight line within the simplex $\mathbf{x}_j\mathbf{x}\mathbf{x}_k$, for sufficiently smooth running costs. Therefore if h is sufficiently small, then $[0, 1]$ will contain a unique minimum and $(0, 0.5, 1)$ will be adequate to determine the minimising abscissa. However, the impact of this bracketing algorithm on the OUM algorithm's run-time also appears to be negligible and, for highly oscillatory running costs in particular, it may provide some advantages by quickly reducing the width of the bracket passed to the Golden Section Search algorithm.

3.3.2 Evaluation

Let $\overline{V}_{\mathbf{yz}}(\mathbf{x}, \zeta)$ be the function in the braces in Equation (4), namely,

$$\overline{V}_{\mathbf{yz}}(\mathbf{x}, \zeta) = \tau(\zeta)R(\mathbf{x}, \boldsymbol{\alpha}_\zeta) + \zeta U(k, l) + (1 - \zeta)U(m, n)\,, \tag{28}$$

where $\mathbf{y} = \mathbf{x}(k, l)$, $\mathbf{z} = \mathbf{x}(m, n)$, $\tau(\zeta) = \|\zeta\mathbf{y} + (1 - \zeta)\mathbf{z} - \mathbf{x}\|$, and the control is given by

$$\boldsymbol{\alpha}_\zeta = \frac{\zeta\mathbf{y} + (1 - \zeta)\mathbf{z} - \mathbf{x}}{\tau(\zeta)}.$$

The notation here is somewhat awkward since U is stored as a function of the mesh coordinates in $\Omega_h^{\mathbb{Z}}$, whereas the other functions in Equation (28) depend on the mesh points in Ω_h.

The evaluation of V in Equation (15) is performed by `TentativeValueFunction`, which takes as inputs h, $\mathbf{x}_{\mathcal{T}}$, $\mathrm{NF}(i, j)$ and $(i, j) \in \mathit{Considered}$, and returns $\{(i, j), V(i, j), \boldsymbol{\alpha}(i, j)\}$; see Algorithm 7. Note that, in practice, lines 8 and 9 of Algorithm 7 would be performed in a single operation.

`TentativeValueFunction` is a major contributor to the total run-time of the OUM algorithm, due to the number of calls to `TentativeValueFunction` and the function minimisation on line 8 of Algorithm 7. Therefore even small improvements to the efficiency of `TentativeValueFunction` have the potential to result in large improvements to the computational performance of the OUM algorithm.

[16] Our algorithm uses parabolic extrapolation to attempt to bracket the minimising abscissa, and if this fails, then the algorithm steps downhill while remaining inside $[0, 1]$ and tries again. If this procedure fails after three attempts, then the last attempt is returned.

Algorithm 7: `TentativeValueFunction`

Input: h, $\mathbf{x}_{\mathcal{T}}$, $\mathrm{NF}(i,j)$, and $(i,j) \in \mathit{Considered}$
Output: $\{(i,j), V(i,j), \boldsymbol{\alpha}(i,j)\}$

if $\mathrm{NF}(i,j) = \emptyset$ **then**
  return $\{(i,j), \infty, \cdot\}$
else
  $\mathbf{x} \leftarrow \texttt{MeshPoint}(h, \mathbf{x}_{\mathcal{T}}, (i,j))$
  $V(i,j) \leftarrow \infty$
  foreach $\{(k,l),(m,n)\} \in \mathrm{NF}(i,j)$ **do**
    $\{\mathbf{y}, \mathbf{z}\} \leftarrow \texttt{MeshPoint}(h, \mathbf{x}_{\mathcal{T}}, \{(k,l),(m,n)\})$
    $C \leftarrow \min_{\zeta \in [0,1]} \overline{V}_{\mathbf{yz}}(\mathbf{x}, \zeta)$
    $\overline{\zeta} \leftarrow \operatorname{argmin}_{\zeta \in [0,1]} \overline{V}_{\mathbf{yz}}(\mathbf{x}, \zeta)$
    if $C < V(i,j)$ **then**
      $V(i,j) \leftarrow C$
      $\boldsymbol{\alpha}(i,j) \leftarrow \boldsymbol{\alpha}_{\overline{\zeta}}$
  return $\{(i,j), V(i,j), \boldsymbol{\alpha}(i,j)\}$

3.4 The OUM Algorithm Revisited

Before revisiting the OUM algorithm, a number of symbols still require definitions. Let $\mathcal{T}_h^{\mathbb{Z}}$ be the set of mesh co-ordinates corresponding to the discretisation of $\mathcal{T}$,

$$\mathcal{T}_h^{\mathbb{Z}} = \big\{(i,j) \in \mathbb{Z}^2 \mid \mathbf{x}(i,j) \in \mathcal{T}_h\big\},$$

noting that $\mathcal{T}_h^{\mathbb{Z}} \subset \Omega_h^{\mathbb{Z}}$. Also let $T(i,j)$ be the triplet

$$T(i,j) = \{(i,j), V(i,j), \boldsymbol{\alpha}(i,j)\},$$

for $(i,j) \in \mathit{Considered}$, and

$$T(i,j) = \{(i,j), U(i,j), \boldsymbol{\alpha}^*(i,j)\},$$

for $(i,j) \in \mathit{Accepted}$.

The OUM algorithm that is the subject of this report is presented in Algorithm 8. The lines that appear as comments in Algorithm 8 correspond to the steps of the OUM algorithm that was reproduced from Sethian & Vladimirsky [2003] in Section 2.4.

Note that

- the running cost $R(\,\cdot\,,\,\cdot\,)$ and anisotropy function $\Upsilon(\,\cdot\,)$ are assumed to be strictly positive Lipschitz-continuous functions; the terminal cost $g(\,\cdot\,) \geqslant 0$ is also assumed to be Lipschitz continuous;
- the dependences of `NearFront` on $\Upsilon(\,\cdot\,)$ and of `TentativeValueFunction` on $R(\,\cdot\,,\,\cdot\,)$ have been suppressed for notational simplicity;
- $\Omega_h^{\mathbb{Z}}$ and $\mathcal{T}_h^{\mathbb{Z}}$ are inputs, and hence must be determined by an algorithm such as Algorithm 1, or by some other means;
- on line 5, the optimal control is not specified on $\mathcal{T}_h^{\mathbb{Z}}$, however it can be specified as a boundary condition on the velocity of the entity;
- the *AcceptedFront* is initialised on the *Accepted* set on line 8;
- the *Considered* set is required to be sorted according to the values of V to determine the minimum on line 14; this can be accomplished via a standard sorting algorithm, such as in Press et al. [1992], or by using a "sort" function included in most mathematical software packages;[17]
- in practice, the solution set S is exported as a data stream on line 15 and stored in a file;
- V is initialised on the new mesh co-ordinates in *Considered* on lines 21 to 25; and
- on lines 26 to 31, `TentativeValueFunction` is called and V is possibly updated to account for the changes to the near front that result from accepting $(\bar{i},\bar{j})$, and therefore Z is a tentative value for V.

[17] Sorting the *Considered* set contributes the $\log(M)$ factor in the computational complexity of the OUM algorithm [Sethian & Vladimirsky 2003].

Algorithm 8: `OrderedUpwindMethod`

Input: $R(\cdot,\cdot)$, $g(\cdot)$, $\Upsilon(\cdot)$, h, $\mathbf{x}_{\mathcal{T}}$, $\Omega_h^{\mathbb{Z}}$, and $\mathcal{T}_h^{\mathbb{Z}}$
Output: solution set S with elements $T(i,j) = \{(i,j), U(i,j), \boldsymbol{\alpha}^*(i,j)\}$ for each $(i,j) \in \Omega_h^{\mathbb{Z}}$

$S \leftarrow \emptyset$
`// step 1`
$Far \leftarrow \Omega_h^{\mathbb{Z}} \setminus \mathcal{T}_h^{\mathbb{Z}}$
`// step 2`
$Accepted \leftarrow \mathcal{T}_h^{\mathbb{Z}}$
foreach $(i,j) \in Accepted$ **do**
  $S \leftarrow S \cup \{\{(i,j), g(\texttt{MeshPoint}(h, \mathbf{x}_{\mathcal{T}}, (i,j))), \cdot\}\}$
`// step 3`
$Considered \leftarrow \texttt{AdjacentTo}(Far, Accepted)$
$Far \leftarrow Far \setminus Considered$
$AcceptedFront \leftarrow Accepted$
$AF \leftarrow \texttt{AFAdjacentPairs}(AcceptedFront, Considered)$
foreach $(i,j) \in Considered$ **do**
  $\mathrm{NF}(i,j) \leftarrow \texttt{NearFront}(h, \mathbf{x}_{\mathcal{T}}, AF, (i,j))$
  $T(i,j) \leftarrow \texttt{TentativeValueFunction}(h, \mathbf{x}_{\mathcal{T}}, \mathrm{NF}(i,j), (i,j))$
while $Considered \neq \emptyset$ **do**
  `// step 4`
  $(\bar{i}, \bar{j}) \leftarrow \operatorname{argmin}_{(i,j) \in Considered} V(i,j)$
  $S \leftarrow S \cup \{T(\bar{i}, \bar{j})\}$
  $N \leftarrow \overline{N}(\bar{i}, \bar{j})$
  `// step 5`
  $Accepted \leftarrow Accepted \cup \{(\bar{i}, \bar{j})\}$
  $Considered \leftarrow Considered \setminus \{(\bar{i}, \bar{j})\}$
  $AcceptedFront \leftarrow (AcceptedFront \setminus N) \cup \texttt{AdjacentTo}(Accepted \cap N, Considered)$
  $AF \leftarrow \texttt{AFUpdate}(AF, AcceptedFront, Considered, N)$
  `// step 6`
  $NewConsidered \leftarrow \texttt{AdjacentTo}(Far, \{(\bar{i}, \bar{j})\})$
  $Far \leftarrow Far \setminus NewConsidered$
  foreach $(i,j) \in NewConsidered$ **do**
    $\mathrm{NF}(i,j) \leftarrow \texttt{NearFront}(h, \mathbf{x}_{\mathcal{T}}, AF, (i,j))$
    $T(i,j) \leftarrow \texttt{TentativeValueFunction}(h, \mathbf{x}_{\mathcal{T}}, \mathrm{NF}(i,j), (i,j))$
  `// step 7`
  $\overline{AF} \leftarrow \texttt{AFSubset}(AF, (\bar{i}, \bar{j}))$
  foreach $(i,j) \in Considered$ **do**
    $\mathrm{NF}(i,j) \leftarrow \texttt{NearFront}(h, \mathbf{x}_{\mathcal{T}}, \overline{AF}, (i,j))$
    $\{(i,j), Z, \boldsymbol{\alpha}(i,j)\} \leftarrow \texttt{TentativeValueFunction}(h, \mathbf{x}_{\mathcal{T}}, \mathrm{NF}(i,j), (i,j))$
    if $Z < V(i,j)$ **then**
      $T(i,j) \leftarrow \{(i,j), Z, \boldsymbol{\alpha}(i,j)\}$
  $Considered \leftarrow Considered \cup NewConsidered$

3.5 Visualisation

`OrderedUpwindMethod` (Algorithm 8) returns $\{(i,j), U(i,j), \boldsymbol{\alpha}^*(i,j)\}$ for each $(i,j) \in \Omega_h^{\mathbb{Z}}$, which is typically stored in a file. This data can then be processed to visualise the solution and determine other properties. One method for visualising the solution is to reconstruct the optimal paths starting at given initial positions to the boundary of the target set. Another visualisation is obtained by generating the level sets of the value function.

3.5.1 Path Reconstruction

Two approaches can be taken to reconstruct the optimal paths from data produced by `OrderedUpwindMethod`. Both commence with a given initial position, and require interpolation to form approximations at points in the domain that are not mesh points. Subsequent points on the optimal path are determined by either

- interpolating the value function data and performing a local minimisation (the value function is strictly monotone decreasing on an optimal path); or
- interpolating the optimal control data and numerically integrating the state equation given by Equation (1).

The second approach is taken in this report.

The algorithm used in this report for reconstructing an optimal path starting at a given initial position $\mathbf{x} \in \Omega$ is as follows:

1. Set the first point on the optimal path $\mathbf{y}_\mathbf{x}$ to be $\mathbf{x}$.
2. Find the vertices of the simplex that contains $\mathbf{y}_\mathbf{x}$.
3. Approximate the optimal control at $\mathbf{y}_\mathbf{x}$ by interpolating the optimal control data on the vertices of the simplex that contains $\mathbf{y}_\mathbf{x}$.
4. Calculate the next point on the optimal path by numerically integrating the state equation, and set $\mathbf{y}_\mathbf{x}$ to be this point.
5. If $\mathbf{y}_\mathbf{x}$ is not within a step size of the target set, then return to step 2 and continue, otherwise return the optimal path.

Step 2 of the path reconstruction algorithm is performed by the `FindSimplex` function, which is defined in Algorithm 9 and motivated by the definition diagram shown in Figure 4.[18] First, let $\mathbf{x}_c = (x_c, y_c) \in \Omega_h$ be the closest mesh point to a given point $\mathbf{x} \in \Omega$, and construct the six candidate simplices with adjacent vertices from the neighbours of $\mathbf{x}_c$. Then determine which quadrant of the (x_c, y_c) co-ordinate system contains $\mathbf{x}$; the number of candidate simplices will now be two. Finally, the simplex that contains $\mathbf{x}$ is determined by considering the cosine of the angle between $\mathbf{x} - \mathbf{x}_c$ and $\mathbf{x}_i - \mathbf{x}_c$, where $i \in \{1, 4\}$, noting that $(\mathbf{x}_{i+1} - \mathbf{x}_c) \cdot (\mathbf{x}_i - \mathbf{x}_c) = 0.5$ for $i \in \{1, \ldots, 5\}$.

[18] `FindSimplex` is a mesh-dependent function, and therefore an implementation of this function for the case of a triangulated mesh constructed from a Cartesian grid has been included in Appendix A.

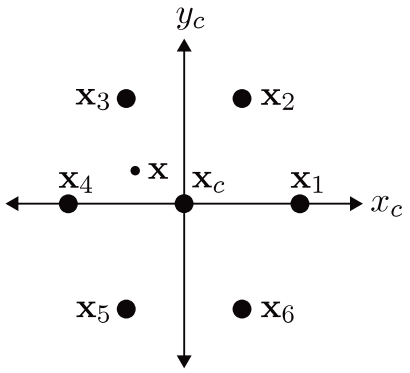


Figure 4: *Definition diagram for the* `FindSimplex` *function (Algorithm 9), showing the neighbours of* $\mathbf{x}_c = (x_c, y_c) \in \Omega_h$ *and an example where the simplex* $\mathbf{x}_c\mathbf{x}_3\mathbf{x}_4$ *contains* $\mathbf{x} \in \Omega$.

An approximation to the optimal control at $\mathbf{x} \in \Omega$ is calculated using linear interpolation. First, `InterpolationCoefficients`[19] (see Algorithm 10) determines the barycentric co-ordinates $\{\zeta_1, \zeta_2, \zeta_3\} \subset \mathbb{R}^3$ of $\mathbf{x}$ with respect to the vertices $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \subset \Omega_h^3$ of the simplex that contains $\mathbf{x}$, by solving the system

$$\zeta_1 + \zeta_2 + \zeta_3 = 1,$$

$$\zeta_1\mathbf{u} + \zeta_2\mathbf{v} + \zeta_3\mathbf{w} = \mathbf{x}.$$

Then step 3 of the path reconstruction algorithm is performed by `InterpolatedControl` (see Algorithm 11), which takes the optimal control data, h, $\mathbf{x}_\mathcal{T}$ and $\mathbf{x}$ as inputs, then calls `FindSimplex` and `InterpolationCoefficients`, returning the linearly interpolated control at $\mathbf{x}$.

Step 4 of the path reconstruction algorithm is performed using Heun's method to numerically integrate Equation (1). Heun's method is a second order Runge-Kutta method, and is often referred to as the improved Euler's method [Leader 2004].

The path reconstruction algorithm is implemented in Algorithm 12, which defines the `OptimalPath` function. `OptimalPath` takes as inputs the optimal control data, $\mathcal{T}_h$, h, $\mathbf{x}_\mathcal{T}$, and the initial position $\mathbf{x} \in \Omega$. `OptimalPath` calls `InterpolatedControl`, numerically integrates Equation (1) using Heun's method with a step size of h, and then returns an ordered list of points on the optimal path with initial position $\mathbf{x}$. Note that

- the distance function $\operatorname{dist}(\,\cdot\,, S) : \Omega \to \mathbb{R}$ (appearing on line 3 of Algorithm 12) is defined to be
$$\operatorname{dist}(\mathbf{x}, S) = \min\{\|\mathbf{x} - \mathbf{y}\| \mid \mathbf{y} \in S\},$$
for a given nonempty finite set $S \subset \Omega$;
- a different numerical method for integrating the state equation can be used by modifying lines 4 to 6 of Algorithm 12; and

[19] `InterpolationCoefficients` is a mesh-dependent function, and therefore an implementation of this function for the case of a triangulated mesh constructed from a Cartesian grid has been included in Appendix A.

Algorithm 9: `FindSimplex`

Input: h, $\mathbf{x}_{\mathcal{T}}$, and $\mathbf{x} \in \Omega$ where $\mathbf{x} = (x_1, x_2)$
Output: $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \subset \Omega_h^3$, where **uvw** is a simplex with adjacent vertices that contains $\mathbf{x}$

$\{\mathbf{x}_c, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\} \leftarrow$ `MeshPoint`$\big(h, \mathbf{x}_{\mathcal{T}}, \overline{N}($`NearestIndex`$(h, \mathbf{x}_{\mathcal{T}}, \mathbf{x}))\big)$
if $x_2 > y_c$ **then**
  if $x_1 > x_c$ **then**
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_1 - \mathbf{x}_c) > 0.5h\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_1, \mathbf{x}_2\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_2, \mathbf{x}_3\}$
  else
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_4 - \mathbf{x}_c) > 0.5h\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_3, \mathbf{x}_4\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_2, \mathbf{x}_3\}$
else
  if $x_1 > x_c$ **then**
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_1 - \mathbf{x}_c) > 0.5h\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_6, \mathbf{x}_1\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_5, \mathbf{x}_6\}$
  else
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_4 - \mathbf{x}_c) > 0.5h\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_4, \mathbf{x}_5\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_5, \mathbf{x}_6\}$

- the discretisation of the target set $\mathcal{T}_h$ can be deduced from the solution data if, for instance, the terminal cost is constant on the boundary of the target set; if this is the case then $\mathcal{T}_h$ is not required as an input.

Algorithm 10: `InterpolationCoefficients`

Input: h, $\mathbf{x} \in \Omega$, and $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \subset \Omega_h^3$, where $\mathbf{uvw}$ is a simplex with adjacent vertices that contains $\mathbf{x}$

Output: $\{\zeta_1, \zeta_2, \zeta_3\} \subset \mathbb{R}^3$, which are barycentric co-ordinates of $\mathbf{x}$ with respect to $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ such that $\zeta_1 + \zeta_2 + \zeta_3 = 1$

$C \leftarrow 2/(3h^2)$
$\mathbf{z} \leftarrow \mathbf{x} - \mathbf{w}$
$C_1 \leftarrow C\mathbf{z} \cdot (\mathbf{u} - \mathbf{v})$
$C_2 \leftarrow C\mathbf{z} \cdot (\mathbf{u} - \mathbf{w})$
$C_3 \leftarrow C\mathbf{z} \cdot (\mathbf{v} - \mathbf{w})$
$\{\zeta_1, \zeta_2, \zeta_3\} \leftarrow \{C_1 + C_2, C_3 - C_1, 1 - C_2 - C_3\}$

Algorithm 11: `InterpolatedControl`

Input: $\boldsymbol{\alpha}^*(i,j)$ for each $(i,j) \in \Omega_h^{\mathbb{Z}}$, h, $\mathbf{x}_{\mathcal{T}}$, and $\mathbf{x} \in \Omega$

Output: a linearly interpolated control $\bar{\boldsymbol{\alpha}}$ at $\mathbf{x}$

$\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \leftarrow$ `FindSimplex`$(h, \mathbf{x}_{\mathcal{T}}, \mathbf{x})$
$\{(i,j), (k,l), (m,n)\} \leftarrow$ `NearestIndex`$(h, \mathbf{x}_{\mathcal{T}}, \{\mathbf{u}, \mathbf{v}, \mathbf{w}\})$
$\{\zeta_1, \zeta_2, \zeta_3\} \leftarrow$ `InterpolationCoefficients`$(h, \mathbf{x}, \{\mathbf{u}, \mathbf{v}, \mathbf{w}\})$
$\bar{\boldsymbol{\alpha}} \leftarrow \zeta_1 \boldsymbol{\alpha}^*(i,j) + \zeta_2 \boldsymbol{\alpha}^*(k,l) + \zeta_3 \boldsymbol{\alpha}^*(m,n)$
$\bar{\boldsymbol{\alpha}} \leftarrow \bar{\boldsymbol{\alpha}} / \|\bar{\boldsymbol{\alpha}}\|$

Algorithm 12: `OptimalPath`

Input: $\boldsymbol{\alpha}^*(i,j)$ for each $(i,j) \in \Omega_h^{\mathbb{Z}}$, $\mathcal{T}_h$, h, $\mathbf{x}_{\mathcal{T}}$, and $\mathbf{x} \in \Omega$

Output: an ordered list L of points on the optimal path with initial position $\mathbf{x}$, that is, $L = \{\mathbf{x}, \mathbf{y}_{\mathbf{x}}(s_1), \mathbf{y}_{\mathbf{x}}(s_2), \dots\}$

$\mathbf{y}_{\mathbf{x}} \leftarrow \mathbf{x}$
$L \leftarrow \{\mathbf{y}_{\mathbf{x}}\}$
while $\operatorname{dist}(\mathbf{y}_{\mathbf{x}}, \mathcal{T}_h) \geqslant h$ **do**
    $\mathbf{k}_1 \leftarrow h$`InterpolatedControl`$(\boldsymbol{\alpha}^*(\cdot\,,\cdot\,), h, \mathbf{x}_{\mathcal{T}}, \mathbf{y}_{\mathbf{x}})$
    $\mathbf{k}_2 \leftarrow h$`InterpolatedControl`$(\boldsymbol{\alpha}^*(\cdot\,,\cdot\,), h, \mathbf{x}_{\mathcal{T}}, \mathbf{y}_{\mathbf{x}} + \mathbf{k}_1)$
    $\mathbf{y}_{\mathbf{x}} \leftarrow \mathbf{y}_{\mathbf{x}} + 0.5(\mathbf{k}_1 + \mathbf{k}_2)$
    $L \leftarrow L \cup \{\mathbf{y}_{\mathbf{x}}\}$

3.5.2 Level Sets

The level sets of u are defined to be $\{\mathbf{x} \in \mathbb{R}^2 \mid u(\mathbf{x}) = C\}$ for a given $C \in \mathbb{R}$. Level sets can be visualised using a "contour" function included in most mathematical software packages. The figures in Section 4 were generated using this approach.

A coarse approximation to the C level set of u can be obtained by monitoring the state of the AF set during the OUM algorithm [Sethian & Vladimirsky 2003, see Remark 7.4]. Let $U^{AF}_{min} = \min_{(i,j)\in AF} U(i,j)$ and $U^{AF}_{max} = \max_{(i,j)\in AF} U(i,j)$, where AF refers to the state of the AF set at the nth iteration and (i,j) denotes an element of a pair of adjacent mesh co-ordinates in AF. If the condition

$$U^{AF}_{min} \leqslant C \leqslant U^{AF}_{max}, \tag{29}$$

is satisfied at the nth iteration of the OUM algorithm, then the corresponding state of AF represents a coarse approximation to the C level set of u. A comparison of this approach with the more accurate "contour" function approach is shown in Figure 9 in Section 4.

4 Examples

The primary aim of this section is to employ `OrderedUpwindMethod` (Algorithm 8) to solve a number of simple path planning problems, to provide figures with which the reader can compare output from their implementation of OUM. This section is not intended to be a demonstration of the utility of OUM for solving real-world problems. Consequently there will only be a very limited discussion of the origin of the running costs used to generate the figures.

In addition to providing the reader with examples, images of the two approaches for visualising level sets that were discussed in Section 3.5.2 are overlaid in Figure 9 for an anisotropic example. To demonstrate the validity of `OrderedUpwindMethod`, the level sets produced by `OrderedUpwindMethod` for another anisotropic running cost and an image of the corresponding figure from Sethian & Vladimirsky [2003] are overlaid in Figure 12.

`OrderedUpwindMethod` was run on a uniform mesh of equilateral triangles on the unit square with $h = 0.00275$ (390^2 mesh points) to generate the figures in this section; an exception is Figure 12, which was generated on a Cartesian grid with spacing $\delta = 0.0026$ ($h = \sqrt{2}\delta$ with 385^2 grid points).[20] In all figures the terminal cost is given by

$$g(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in \partial\mathcal{T} \\ \infty, & \text{otherwise,} \end{cases}$$

where $\partial\mathcal{T}$ is the boundary of the target set; this ensures that all optimal paths hit the target set.[21] The target set is chosen to be the singleton $\{\mathbf{x}_\mathcal{T}\}$, which is approximated by the neighbours of $\mathbf{x}_\mathcal{T}$.

[20] Correspondence with A. Vladimirsky revealed that Figure 5 from Sethian & Vladimirsky [2003] was generated on a Cartesian grid.

[21] Theoretically, this terminal cost is required to be mollified. However, in practice, the value function is simply set to zero on the boundary of the target set.

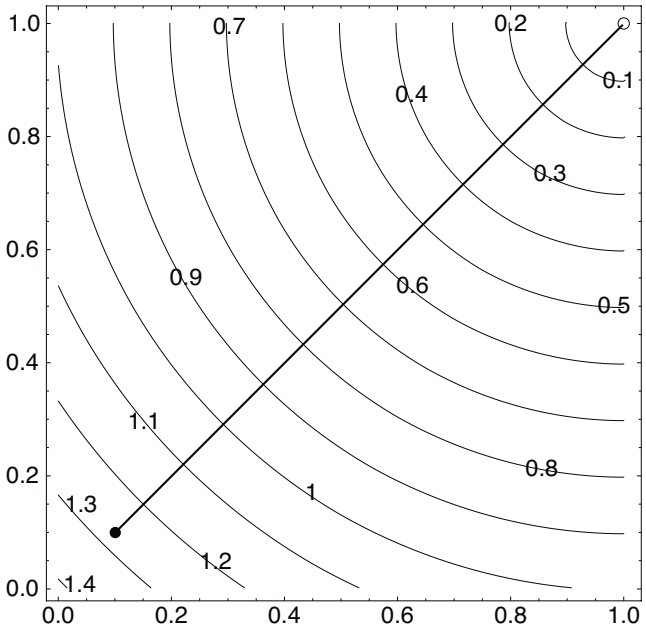


Figure 5: *Uniform running cost ($R = 1$), showing level sets and an optimal path with initial position $\mathbf{x} = (0.1, 0.1)$ and target location $\mathbf{x}_T = (1, 1)$.*

4.1 Isotropic Running Costs

The first isotropic example is the trivial case of a uniform running cost, given by $R(\mathbf{x}) = 1$ with anisotropy function $\Upsilon(\mathbf{x}) = 1$. In Figure 5 it can be seen that, as expected, the value function is simply the distance to the boundary of the target set, the level sets are concentric circles centred at the target, and the optimal paths are the shortest straight-line paths from given initial positions to the boundary of the target set.

The second isotropic example is an obstacle avoidance problem taken from Kumar & Vladimirsky [2010], and is displayed in Figures 6 and 7. The obstacles are three squares with side length $L_s = 0.1$, located at $\mathbf{s}_1 = (0.25, 0.5)$, $\mathbf{s}_2 = (0.5, 0.3)$ and $\mathbf{s}_3 = (0.65, 0.75)$. Let the shortest distance to the obstacles be

$$\mathcal{O}(\mathbf{x}) = \min\left\{\|\mathbf{x} - \mathbf{s}_1\|_\infty, \|\mathbf{x} - \mathbf{s}_2\|_\infty, \|\mathbf{x} - \mathbf{s}_3\|_\infty\right\}.$$

The running cost is given by[22]

$$R(\mathbf{x}) = \begin{cases} \dfrac{1}{1 + 0.8\sin(4\pi x_1)\sin(6\pi x_2)}, & \mathcal{O}(\mathbf{x}) > 0.5L_s \\ \infty, & \text{otherwise,} \end{cases} \tag{30}$$

and the anisotropy function is $\Upsilon(\mathbf{x}) = 1$.

Observe in Figures 5 and 6 that the optimal paths are orthogonal to the levels sets of the value function, as expected for isotropic running costs.

[22]This running cost should be mollified to be consistent with the theory that underpins OUM. Alternatively, the solution can be interpreted in terms of a discontinuous value function [Bardi & Capuzzo-Dolcetta 2008].

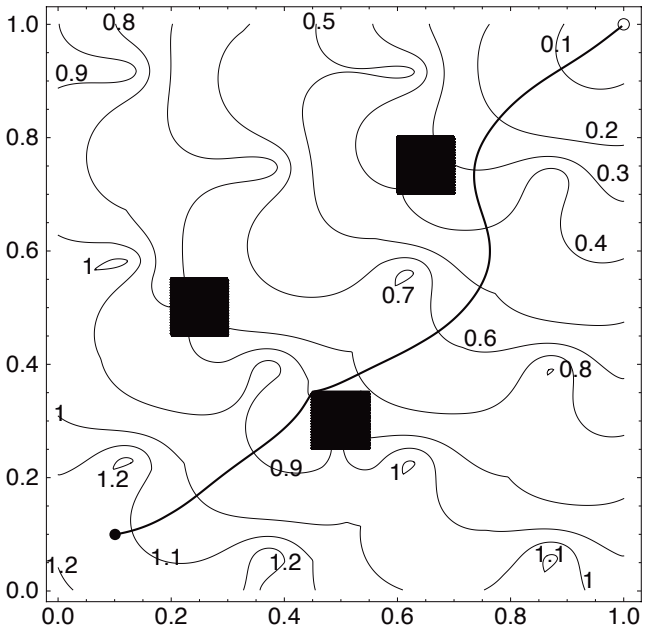


Figure 6: *Isotropic running cost (given by Equation (30)), showing level sets and an optimal path with initial position* $\mathbf{x} = (0.1, 0.1)$ *and target location* $\mathbf{x}_{\mathcal{T}} = (1, 1)$*. The black squares represent obstacles, which have a side length of* 0.1 *and are located at* $(0.25, 0.5)$*,* $(0.5, 0.3)$ *and* $(0.65, 0.75)$*.*

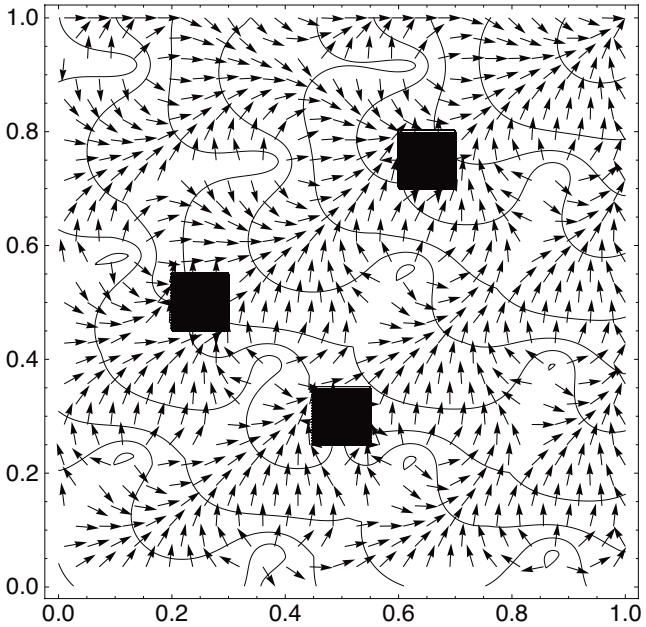


Figure 7: *Isotropic running cost (given by Equation (30)), showing level sets and a representation of a subset of the optimal controls. The target location is* $\mathbf{x}_{\mathcal{T}} = (1, 1)$ *and the* $\{0.1, 0.2, \ldots, 1.2\}$ *level sets are displayed.*

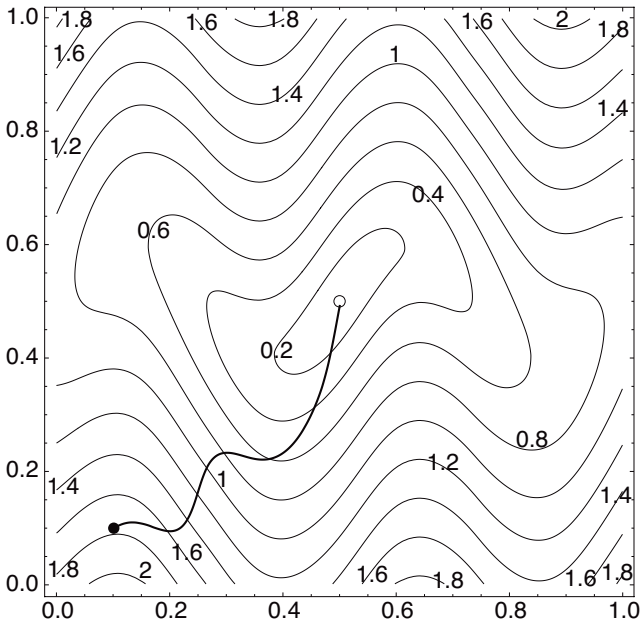


Figure 8: *Anisotropic running cost (given by Equation (31)), showing level sets and an optimal path with initial position* $\mathbf{x} = (0.1, 0.1)$ *and target location* $\mathbf{x}_{\mathcal{T}} = (0.5, 0.5)$.

4.2 Anisotropic Running Costs

The first anisotropic example originated as a seismic imaging problem, and can be found in Sethian & Vladimirsky [2003] and Kumar & Vladimirsky [2010]. The running cost is given by

$$R(\mathbf{x}, \boldsymbol{\alpha}) = 1.25\sqrt{1 + \left(\sqrt{\frac{15}{1 + (0.49\pi\cos(4\pi x_1))^2}}\,(0.49\pi\cos(4\pi x_1), -1) \cdot \boldsymbol{\alpha}\right)^2}, \qquad (31)$$

and the anisotropy function is $\Upsilon(\mathbf{x}) = 4$. The solution to this seismic imaging problem is visualised in Figure 8.[23]

Images of the two approaches for visualising level sets that were discussed in Section 3.5.2 are overlaid in Figure 9 for this anisotropic example (Equation (31)). Note that raw *AF* data (line segments) obtained by applying the condition given by Equation (29) is shown in Figure 9, together with the level sets generated by a "contour" function. The excellent visual agreement that can be observed in Figure 9 is primarily due to the relatively fine mesh used to produce the solution data; for a more coarse mesh, differences between these approaches are more noticeable.

[23]There are small qualitative differences between Figure 8 and the corresponding figure in Kumar & Vladimirsky [2010], primarily in the optimal path. Kumar & Vladimirsky adapt OUM to solve constrained and multiobjective optimal control problems. For unconstrained (and single objective) optimal control problems their numerical method coincides with OUM in the limit as $h \to 0$. However, for a given mesh with h away from zero, some differences in the solutions produced using these methods is to be expected. Furthermore, the corresponding figure in Kumar & Vladimirsky [2010] was generated on a Cartesian grid

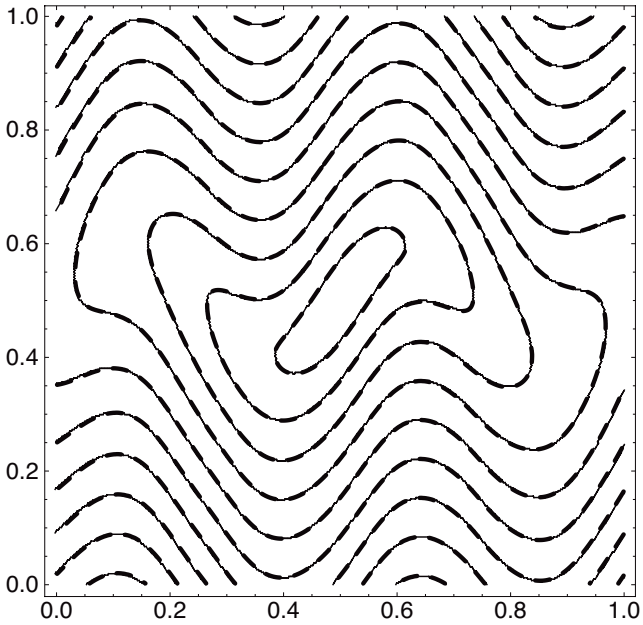


Figure 9: *Anisotropic running cost (given by Equation (31)), showing level sets generated by a "contour" function (thick dashed curves) and the level sets generated using Equation (29) (thin solid curves). The* $\{0.2, 0.4, \ldots, 2.0\}$ *level sets are displayed.*

The final anisotropic example is related to the geodesic distance on a manifold, and comes from Sethian & Vladimirsky [2003]. The running cost is

$$R(\mathbf{x}, \boldsymbol{\alpha}) = \sqrt{1 + \left(1.8\pi \left(\cos(2\pi x_1)\sin(2\pi x_2), \cos(2\pi x_2)\sin(2\pi x_1)\right) \cdot \boldsymbol{\alpha}\right)^2}, \tag{32}$$

and the anisotropy function is $\Upsilon(\mathbf{x}) = 5.75$. The solution to this example is visualised in Figures 10 and 11.

Observe in Figures 8 and 10 that the optimal paths are not necessarily orthogonal to the levels sets of the value function, as expected for anisotropic running costs.

To demonstrate the validity of `OrderedUpwindMethod`, the level sets produced by `OrderedUpwindMethod` for this anisotropic example (Equation (32)) and an image of the corresponding figure from Sethian & Vladimirsky [2003] are overlaid in Figure 12, where excellent visual agreement can be observed.

with 201^2 grid points, whereas Figure 8 was generated on a uniform mesh of equilateral triangles with 390^2 mesh points.

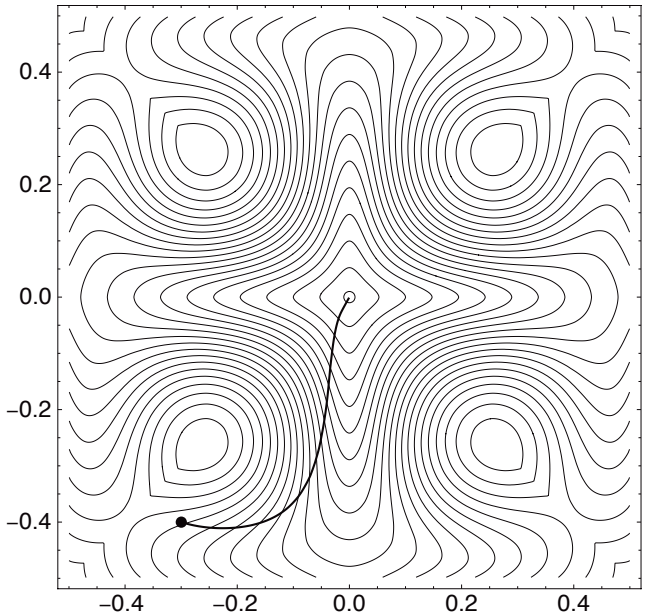


Figure 10: *Anisotropic running cost (given by Equation (32)), showing level sets and an optimal path with initial position* $\mathbf{x} = (-0.3, -0.4)$ *and target location* $\mathbf{x}_{\mathcal{T}} = (0, 0)$*. The* $\{0.05, 0.0973684, \ldots, 0.95\}$ *level sets are displayed.*

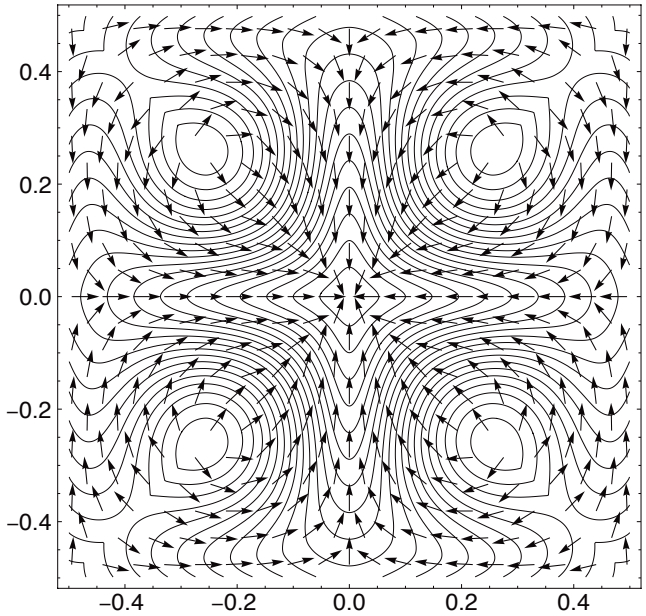


Figure 11: *Anisotropic running cost (given by Equation (32)), showing level sets and a representation of a subset of the optimal controls. The target location is* $\mathbf{x}_{\mathcal{T}} = (0, 0)$ *and the* $\{0.05, 0.0973684, \ldots, 0.95\}$ *level sets are displayed.*

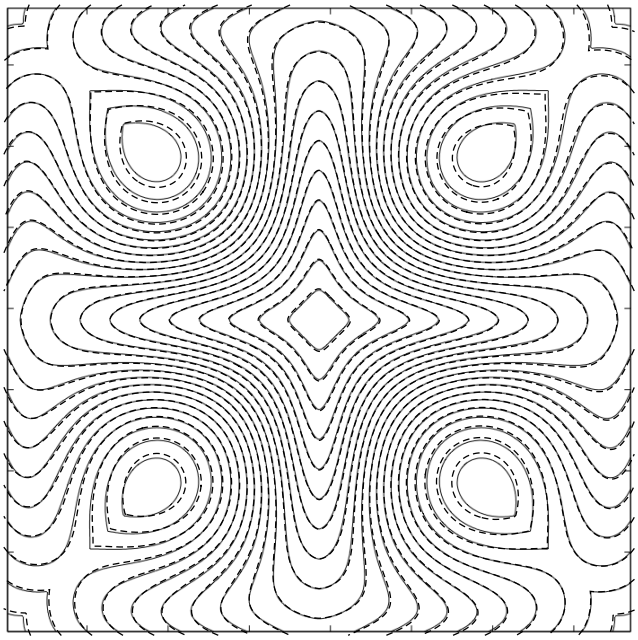


Figure 12: *Anisotropic running cost (given by Equation (32)), showing level sets generated by* `OrderedUpwindMethod` *(dashed curves) and the level sets reproduced from Figure 5 of Sethian & Vladimirsky [2003] (solid curves); both sets of contours were generated on a* 385^2 *Cartesian grid.*

5 Conclusion

There are many diverse numerical methods that can be applied to solving path planning problems, however most of these are either not valid or impractical for solving anisotropic path planning problems. Ordered Upwind Methods (OUM) are a family of numerical methods for approximating the viscosity solution of static Hamilton-Jacobi-Bellman (HJB) equations, and have been tailored to solve anisotropic optimal control problems. The focus of this report has been on the control-theoretic OUM.

There is little information in the literature regarding the implementation of OUM, and a wide range of computational techniques and meticulous algorithmic considerations are required to successfully implement OUM. A comprehensive, generic implementation of OUM has been documented in Section 3 of this report, with the intention of minimising the technical barriers to employing OUM in real-world applications.

The OUM algorithm processes subsets of the computational domain to compute the numerical value function. The run-time of the OUM algorithm can be significantly reduced by noting that elements are only added to (or removed from) the *AcceptedFront* from a neighbourhood of the most recently accepted mesh co-ordinate. This is exploited in the *AcceptedFront* update rule given by Equation (24) and in the definition of the `AFUpdate` function (Algorithm 3).

Note that the major contributors to the run-time of the OUM algorithm are near front construction (refer to Algorithms 5 and 6) and the evaluation of the tentative value

function (Algorithm 7). This is caused by the recalculation of the tentative value function to account for the changes to the near front that result from accepting a mesh co-ordinate at each iteration of the OUM algorithm; refer to step 7 of Algorithm 8.

If the anisotropy function satisfies $\Upsilon(\mathbf{x}) \gg 1/h$ on a significant subset of the domain, then $\mathrm{NF}(\mathbf{x}) \approx AF$ and hence the run-time of OUM may considerably increase. Decreasing the run-time of OUM typically requires solving the HJB equation on a non-uniform mesh. One such approach entails refining the mesh on subsets of the domain where $\Upsilon(\mathbf{x}) \gg 1/h$. This mesh refinement condition, together with the bisection refinement algorithm of Maubach [1995] and the data structures presented in Cline & Renka [1984], have been implemented by the author and the initial results are promising. Other methods require two passes through the domain to construct the numerical solution of the HJB equation. The so-called Patchy Dynamic Programming scheme of Cacace et al. [2012] requires the HJB equation to be solved initially on a coarse mesh to determine the subsets of the domain, known as patches, where the solution can be independently computed in the second stage of the scheme. The HJB equation is then solved using a finer mesh on each patch utilizing parallel computing. Alton & Mitchell [2012] have developed a variant of the OUM that is optimised for highly non-uniform meshes. For this two-pass approach, the computational stencil is first computed for each node such that during the second pass, the HJB equation can be solved using a fast Dijkstra-like method where the nodes are accepted monotonically.

The aforementioned techniques for decreasing the run-time of OUM when $\Upsilon(\mathbf{x}) \gg 1/h$ continue to be investigated by the author. Applying OUM to the path planning of military aircraft traversing hostile environments will also be studied in future work.

Acknowledgements

During the preparation of this report I received invaluable feedback from Maria Athanassenas, Russell Connell, David Cox, Emily Duane, Simon Goss, Michael Papasimeon, Olivia Smith and Josef Zuk; this work would not have come to fruition without their input. In particular, I express my thanks to Josef Zuk for suggesting an approach that led to Algorithm 9, and Alexander Vladimirsky for his valuable correspondence.

References

Alton, K. & Mitchell, I. M. (2012) An Ordered Upwind Method with precomputed stencil and monotone node acceptance for solving static convex Hamilton-Jacobi equations, *Journal of Scientific Computing* **51**(2), 313–348.

Arnold, D. N., Mukherjee, A. & Pouly, L. (2000) Locally adapted tetrahedral meshes using bisection, *SIAM J. Sci. Comput.* **22**(2), 431–448.

Bänsch, E. (1991) Local mesh refinement in 2 and 3 dimensions, *Impact of Computing in Science and Engineering* **3**, 181–191.

Bardi, M. & Capuzzo-Dolcetta, I. (2008) *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*, Modern Birkhäuser Classics, Birkhäuser, Boston.

Beard, R. W., McLain, T. W., Goodrich, M. A. & Anderson, E. P. (2002) Coordinated target assignment and intercept for unmanned air vehicles, *IEEE Transactions on Robotics and Automation* **18**(6), 911–922.

Betts, J. T. (1998) Survey of numerical methods for trajectory optimization, *Journal of Guidance, Control, and Dynamics* **21**(2), 193–207.

Bey, J. (1995) Tetrahedral grid refinement, *Computing* **55**, 355–378.

Bortoff, S. A. (2000) Path planning for UAVs, *in Proceedings of the American Control Conference*, pp. 364–368.

Cacace, S., Cristiani, E., Falcone, M. & Picarelli, A. (2012) A Patchy Dynamic Programming scheme for a class of Hamilton–Jacobi–Bellman equations, *SIAM J. Sci. Comput.* **34**(5), A2625—A2649.

Chaudhry, A., Misovec, K. & D'Andrea, R. (2004) Low observability path planning for an unmanned air vehicle using mixed integer linear programming, *in 43rd IEEE Conference on Decision and Control*, pp. 3823–3829.

Cline, A. K. & Renka, R. L. (1984) A storage-efficient method for construction of a Thiessen triangulation, *Rocky Mountain Journal of Mathematics* **14**(1), 119–139.

Cristiani, E. (2009) A fast marching method for Hamilton-Jacobi equations modeling monotone front propagations, *Journal of Scientific Computing* **39**, 189–205.

Dai, R. & Cochran Jr., J. E. (2010) Path planning and state estimation for unmanned aerial vehicles in hostile environments, *Journal of Guidance, Control, and Dynamics* **33**(2), 595–601.

Dijkstra, E. W. (1959) A note on two problems in connexion with graphs, *Numerische Mathematik* **1**, 269–271.

Evans, L. C. (1998) *Partial Differential Equations*, Vol. 19 of *Graduate Studies in Mathematics*, American Mathematical Society, Providence, R.I.

Hjelle, Ø. & Dæhlen, M. (2006) *Triangulations and Applications*, Mathematics and Visualization, Springer-Verlag, Berlin.

Hwang, Y. K. & Ahuja, N. (1992) Gross motion planning—a survey, *ACM Computing Surveys* **24**(3), 219–291.

Inanc, T., Muezzinoglu, M. K., Misovec, K. & Murray, R. M. (2008) Framework for low-observable trajectory generation in presence of multiple radars, *Journal of Guidance, Control, and Dynamics* **31**(6), 1740–1749.

Kabamba, P. T., Meerkov, S. M. & Zeitz III, F. H. (2006) Optimal path planning for unmanned combat aerial vehicles to defeat radar tracking, *Journal of Guidance, Control, and Dynamics* **29**(2), 279–288.

Kao, C.-Y., Osher, S. & Tsai, Y.-H. (2005) Fast sweeping methods for static Hamilton-Jacobi equations, *SIAM J. Numer. Anal.* **42**(6), 2612–2632.

Kim, J. & Hespanha, J. P. (2003) Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of uavs, *in 42nd IEEE Conference on Decision and Control*, pp. 1734–1740.

Kumar, A. & Vladimirsky, A. (2010) An efficient method for multiobjective optimal control and optimal control subject to integral constraints, *Journal of Computational Mathematics* **28**(4), 517–551.

LaValle, S. M. & Kuffner, Jr., J. J. (2001) Randomized kinodynamic planning, *The International Journal of Robotics Research* **20**, 378–400.

Leader, J. J. (2004) *Numerical Analysis and Scientific Computation*, Addison Wesley, Boston.

Maubach, J. M. (1995) Local bisection refinement for n-simplicial grids generated by reflection, *SIAM J. Sci. Comput.* **16**(1), 210–227.

McLain, T. W. & Beard, R. W. (2005) Coordination variables, coordination functions, and cooperative-timing missions, *Journal of Guidance, Control, and Dynamics* **28**(1), 150–161.

Mercer, G. N. & Sidhu, H. S. (2007) Two continuous methods for determining a minimal risk path through a minefield, *ANZIAM J.* **48**, C293–C306.

Milam, M. B., Mushambi, K. & Murray, R. M. (2000) A new computational approach to real-time trajectory generation for constrained mechanical systems, *in Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 845–851.

Mitchell, I. M. & Sastry, S. (2003) Continuous path planning continuous path planning with multiple constraints, *in Proceedings of the 42nd IEEE Conference on Decision and Control*, pp. 5502–5507.

Muhandiramge, R., Boland, N. & Wang, S. (2009) Convergent network approximation for the continuous euclidean length constrained minimum cost path problem, *SIAM Journal on Optimization* **20**(1), 54—77.

Novy, M. C., Jacques, D. R. & Pachter, M. (2002) Air vehicle optimal trajectories between two radars, *in Proceedings of the American Control Conference*.

Pachter, M. & Hebert, J. (2001) Optimal aircraft trajectories for radar exposure minimization, *in Proceedings of the American Control Conference*.

Pêtrès, C., Pailhas, Y., Patrón, P., Petillot, Y., Evans, J. & Lane, D. (2007) Path planning for autonomous underwater vehicles, *IEEE Transactions on Robotics* **23**(2), 331–341.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992) *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn, Cambridge University Press.

Qian, J., Zhang, Y.-T. & Zhao, H.-K. (2007*a*) A fast sweeping method for static convex Hamilton–Jacobi equations, *Journal of Scientific Computing* **31**(1/2), 237–271.

Qian, J., Zhang, Y.-T. & Zhao, H.-K. (2007*b*) Fast sweeping methods for Eikonal equations on triangular meshes, *SIAM J. Numer. Anal.* **45**(1), 83—107.

Rowe, M. P., Sidhu, H. S. & Mercer, G. N. (2009) Military aviation applications for a springs and masses safest path determining model, *Journal of Battlefield Technology* **12**(1), 27–32.

Ruppert, J. (1995) A Delaunay refinement algorithm for quality 2-dimensional mesh generation, *Journal of Algorithms* **18**, 548–585.

Sethian, J. A. (1999*a*) Fast marching methods, *SIAM Review* **41**(2), 199–235.

Sethian, J. A. (1999*b*) *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge Monographs on Applied and Computational Mathematics, 2nd edn, Cambridge University Press, Cambridge.

Sethian, J. A. & Vladimirsky, A. (2001) Ordered upwind methods for static Hamilton–Jacobi equations, *Proceedings of the National Academy of Sciences of the USA* **98**(20), 11069—11074.

Sethian, J. A. & Vladimirsky, A. (2003) Ordered upwind methods for static Hamilton–Jacobi equations: theory and algorithms, *SIAM J. Numer. Anal.* **41**(1), 325–363.

Shewchuk, J. R. (2002) Delaunay refinement algorithms for triangular mesh generation, *Computational Geometry* **22**, 21–74.

Sidhu, H. S., Mercer, G. N., Sexton, M. J., Ansari, N. A. & Jovanoski, Z. (2006) Optimal path trajectories in a threat environment, *Journal of Battlefield Technology* **9**(3), 1–7.

Tsai, Y.-H. R., Cheng, L.-T., Osher, S. & Zhao, H.-K. (2003) Fast sweeping algorithms for a class of Hamilton–Jacobi equations, *SIAM J. Numer. Anal.* **41**(2), 673–694.

Tsitsiklis, J. N. (1995) Efficient algorithms for globally optimal trajectories, *IEEE Transactions on Automatic Control* **40**(9).

Vian, J. L. & Moore, J. R. (1989) Trajectory optimization with risk minimization for military aircraft, *Journal of Guidance* **12**(3), 311–317.

Vladimirsky, A. B. (2001) *Fast methods for static Hamilton-Jacobi Partial Differential Equations*, PhD thesis, University of California, Lawrence Berkeley National Laboratory.

Zabarankin, M., Uryasev, S. & Murphey, R. (2006) Aircraft routing under the risk of detection, *Naval Research Logistics* **53**, 728–747.

Zheng, C., Li, L., Xu, F., Sun, F. & Ding, M. (2005) Evolutionary route planner for unmanned air vehicles, *IEEE Transactions on Robotics* **21**(4), 609–620.

THIS PAGE IS INTENTIONALLY BLANK

Appendix A Triangulated Mesh Based on a Cartesian Grid

An adaption of the key mesh dependent functions for the case of a triangulated mesh based on a Cartesian grid is presented in this appendix.

The mesh point $\mathbf{x}(i,j) \in \Omega_h$ indexed by the mesh co-ordinate $(i,j) \in \Omega_h^{\mathbb{Z}}$ is given by

$$\mathbf{x}(i,j) = \mathbf{x}_{\mathcal{T}} + \delta(i,j)\,, \tag{A1}$$

where δ is the grid spacing and the triangulation diameter $h = \sqrt{2}\delta$. The functions for mesh transformations and mesh construction follow directly from Equation (A1).

There are many ways of constructing a triangulated mesh from a Cartesian grid, and a general discussion of this topic is beyond the scope of this report. Instead, the set of neighbours $N(i,j)$ which has the same mesh co-ordinates as for a uniform mesh of equilateral triangles (see Equation (20)) is shown in Figure A1.

The modifications of the `FindSimplex` and `InterpolationCoefficients` functions for the case of a Cartesian grid can be found in Algorithms 13 and 14, respectively.

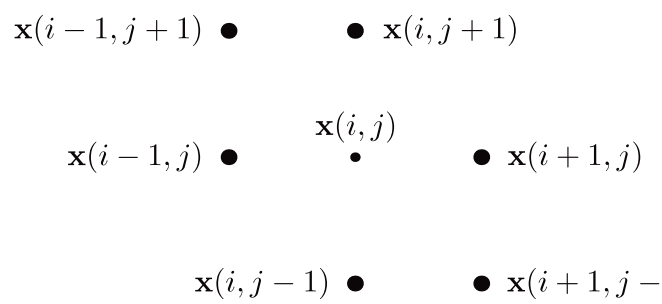


Figure A1: *The $N(i,j)$ set shown mapped onto Ω_h as large dots for the case of a triangulation based on a Cartesian grid. The $\mathbf{x}(\,\cdot\,,\,\cdot\,)$ are given by Equation (A1) and the triangulation diameter $h = \sqrt{2}\delta$, where δ is the grid spacing.*

Algorithm 13: `FindSimplex` for the case of a Cartesian grid

Input: δ, $\mathbf{x}_{\mathcal{T}}$, and $\mathbf{x} \in \Omega$ where $\mathbf{x} = (x_1, x_2)$
Output: $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \subset \Omega_h^3$, where **uvw** is a simplex with adjacent vertices that contains $\mathbf{x}$

$\{\mathbf{x}_c, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\} \leftarrow \texttt{MeshPoint}\big(\delta, \mathbf{x}_{\mathcal{T}}, \overline{N}(\texttt{NearestIndex}(\delta, \mathbf{x}_{\mathcal{T}}, \mathbf{x}))\big)$
if $x_2 > y_c$ **then**
  if $x_1 > x_c$ **then**
    return $\{\mathbf{x}_c, \mathbf{x}_1, \mathbf{x}_2\}$
  else
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_4 - \mathbf{x}_c) > \sqrt{0.5}\delta\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_3, \mathbf{x}_4\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_2, \mathbf{x}_3\}$
else
  if $x_1 < x_c$ **then**
    return $\{\mathbf{x}_c, \mathbf{x}_4, \mathbf{x}_5\}$
  else
    if $(\mathbf{x} - \mathbf{x}_c) \cdot (\mathbf{x}_1 - \mathbf{x}_c) > \sqrt{0.5}\delta\|\mathbf{x} - \mathbf{x}_c\|$ **then**
      return $\{\mathbf{x}_c, \mathbf{x}_6, \mathbf{x}_1\}$
    else
      return $\{\mathbf{x}_c, \mathbf{x}_5, \mathbf{x}_6\}$

Algorithm 14: `InterpolationCoefficients` for the case of a Cartesian grid

Input: δ, $\mathbf{x} \in \Omega$, and $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} \subset \Omega_h^3$, where **uvw** is a simplex with adjacent vertices that contains $\mathbf{x}$
Output: $\{\zeta_1, \zeta_2, \zeta_3\} \subset \mathbb{R}^3$, which are barycentric co-ordinates of $\mathbf{x}$ with respect to $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ such that $\zeta_1 + \zeta_2 + \zeta_3 = 1$

$C \leftarrow 1/\delta^2$
$\mathbf{z} \leftarrow \mathbf{x} - \mathbf{w}$
$C_1 \leftarrow C\mathbf{z} \cdot (\mathbf{u} - \mathbf{w})$
$C_2 \leftarrow C\mathbf{z} \cdot (\mathbf{v} - \mathbf{w})$
$\{\zeta_1, \zeta_2, \zeta_3\} \leftarrow \{C_1, C_2, 1 - C_1 - C_2\}$

Appendix B Verification of the *AcceptedFront* update rule

The aim of this appendix is to verify the *AcceptedFront* update rule given by Equation (24).

To begin, let the states of the *Accepted* and *Considered* sets that are used to update the *AcceptedFront* during the nth iteration of the OUM algorithm be given by A_n and C_n, respectively. Then Equation (24) can be expressed as

$$\mathit{AcceptedFront}_n = \big(\mathit{AcceptedFront}_{n-1} \setminus \overline{N}(\bar{\mathbf{x}}_n)\big) \cup \texttt{AdjacentTo}(A_n \cap \overline{N}(\bar{\mathbf{x}}_n), C_n), \quad \text{(B1)}$$

where $\bar{\mathbf{x}}_n$ is the most recently accepted mesh point at the nth iteration.[24] Recall that, by definition,

$$\mathit{AcceptedFront}_n = \texttt{AdjacentTo}(A_n, C_n). \quad \text{(B2)}$$

The remainder of this appendix is devoted to proving that Equations (B1) and (B2) generate the same sets.

For any sets X and Y, $X = (X \setminus Y) \cup (X \cap Y)$. Therefore

$$\mathit{AcceptedFront}_n = \big(\mathit{AcceptedFront}_n \setminus \overline{N}(\bar{\mathbf{x}}_n)\big) \cup \texttt{AdjacentTo}(A_n \cap \overline{N}(\bar{\mathbf{x}}_n), C_n),$$

by Equation (B2) and the definition of `AdjacentTo` in Equation (22). Consequently, if

$$\mathit{AcceptedFront}_n \setminus \overline{N}(\bar{\mathbf{x}}_n) = \mathit{AcceptedFront}_{n-1} \setminus \overline{N}(\bar{\mathbf{x}}_n),$$

then Equations (B1) and (B2) will generate the same sets. Observe that[25]

$$\begin{aligned} \mathit{AcceptedFront}_n \setminus \overline{N}(\bar{\mathbf{x}}_n) &= A_n \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= (A_{n-1} \cup \{\bar{\mathbf{x}}_n\}) \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= A_{n-1} \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c, \end{aligned}$$

as $A_n = A_{n-1} \cup \{\bar{\mathbf{x}}_n\}$ and $\{\bar{\mathbf{x}}_n\} \cap \overline{N}(\bar{\mathbf{x}}_n)^c = \emptyset$. Furthermore,

$$\mathit{AcceptedFront}_{n-1} \setminus \overline{N}(\bar{\mathbf{x}}_n) = A_{n-1} \cap N(C_{n-1}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c.$$

Therefore if

$$A_{n-1} \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c = A_{n-1} \cap N(C_{n-1}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c, \quad \text{(B3)}$$

then Equations (B1) and (B2) will generate the same sets.

To establish Equation (B3), the following identity is required for sets X, $Y \subset \Omega_h^{\mathbb{Z}}$:

$$N(X \cup Y) = (N(X) \cup N(Y)) \setminus \texttt{AdjacentTo}(X, Y), \quad \text{if } X \cap Y = \emptyset. \quad \text{(B4)}$$

Let $X \cap Y = \emptyset$. If X, Y also satisfy $\texttt{AdjacentTo}(X, Y) = \emptyset$, then $N(X \cup Y) = N(X) \cup N(Y)$. However if $\texttt{AdjacentTo}(X, Y) \neq \emptyset$, then $N(X)$ will contain elements that belong to Y,

[24] Here it is more convenient to work with mesh points instead of mesh co-ordinates, which are used in Equation (24).

[25] Recall that $X \setminus Y = X \cap Y^c$, where Y^c is the complement of Y.

and $N(Y)$ will contain elements of X. Equation (B4) follows from these observations, noting that $\texttt{AdjacentTo}(X, Y) = \texttt{AdjacentTo}(Y, X)$ provided $X \cap Y = \emptyset$.

The *Considered* set at the nth iteration of the OUM algorithm can be related to C_{n-1} via

$$C_n \cup \{\bar{\mathbf{x}}_n\} = C_{n-1} \cup C_n^{new},$$

where $C_n^{new} \subseteq C_n$ is the set of new points added to *Considered* during the previous iteration. Consequently

$$N(C_n \cup \{\bar{\mathbf{x}}_n\}) = N(C_{n-1} \cup C_n^{new}). \tag{B5}$$

Continuing,[26]

$$\begin{aligned} N(C_n \cup \{\bar{\mathbf{x}}_n\}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c &= (N(C_n) \cup N(\bar{\mathbf{x}}_n)) \cap (C_n \cap N(\bar{\mathbf{x}}_n))^c \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= (N(C_n) \cup N(\bar{\mathbf{x}}_n)) \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c, \end{aligned}$$

by Equation (B4), noting that $C_n \cap N(\bar{\mathbf{x}}_n) \subset \overline{N}(\bar{\mathbf{x}}_n)$ and $N(\bar{\mathbf{x}}_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c = \emptyset$. Therefore

$$A_{n-1} \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c = A_{n-1} \cap N(C_n \cup \{\bar{\mathbf{x}}_n\}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c. \tag{B6}$$

Similarly,[27]

$$\begin{aligned} A_{n-1} \cap N(C_{n-1} \cup C_n^{new}) &= A_{n-1} \cap (N(C_{n-1}) \cup N(C_n^{new})) \cap (C_{n-1} \cap N(C_n^{new}))^c \\ &= A_{n-1} \cap \left(C_{n-1}^c \cup N(C_n^{new})^c\right) \cap (N(C_{n-1}) \cup N(C_n^{new})) \\ &= \left(\left(A_{n-1} \cap C_{n-1}^c\right) \cup (A_{n-1} \cap N(C_n^{new})^c)\right) \cap (N(C_{n-1}) \cup N(C_n^{new})) \\ &= A_{n-1} \cap (N(C_{n-1}) \cup N(C_n^{new})) \\ &= (A_{n-1} \cap N(C_{n-1})) \cup (A_{n-1} \cap N(C_n^{new})) \\ &= A_{n-1} \cap N(C_{n-1}), \end{aligned}$$

using Equation (B4), noting that $C_{n-1}^c = A_{n-1} \cup Far_{n-1}$, $A_{n-1} \subset A_{n-1} \cap N(C_n^{new})^c$, and $A_{n-1} \cap N(C_n^{new}) = \texttt{AdjacentTo}(A_{n-1}, C_n^{new}) = \emptyset$, as $C_n^{new} \subseteq Far_{n-1}$. It follows that

$$A_{n-1} \cap N(C_{n-1}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c = A_{n-1} \cap N(C_{n-1} \cup C_n^{new}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c. \tag{B7}$$

Combining Equations (B5) to (B7) yields

$$\begin{aligned} A_{n-1} \cap N(C_n) \cap \overline{N}(\bar{\mathbf{x}}_n)^c &= A_{n-1} \cap N(C_n \cup \{\bar{\mathbf{x}}_n\}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= A_{n-1} \cap N(C_{n-1} \cup C_n^{new}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c \\ &= A_{n-1} \cap N(C_{n-1}) \cap \overline{N}(\bar{\mathbf{x}}_n)^c. \end{aligned}$$

Therefore Equation (B3) is verified, and hence Equations (B1) and (B2) generate the same sets.

[26] Recall that $(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z)$, and $(X \cap Y) \cup Z = (X \cup Z) \cap (Y \cup Z)$.

[27] De Morgan's Laws state that $(X \cup Y)^c = X^c \cap Y^c$ and $(X \cap Y)^c = X^c \cup Y^c$.

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE	3. SECURITY CLASSIFICATION
Globally Optimal Path Planning with Anisotropic Running Costs	Document (U) Title (U) Abstract (U)
4. AUTHOR	5. CORPORATE AUTHOR
Jason R. Looker	Defence Science and Technology Organisation 506 Lorimer St, Fishermans Bend, Victoria 3207, Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO–TR–2815	AR 015–556	Technical Report	March, 2013

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. No. OF PAGES	12. No. OF REFS
2012/1105822/1	DS 07/245	COAD	42	50

13. URL OF ELECTRONIC VERSION	14. RELEASE AUTHORITY
`http://www.dsto.defence.gov.au/publications/scientific.php`	Chief, Air Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DSTO RESEARCH LIBRARY THESAURUS

collision avoidance, mission planning, numerical algorithms, optimal control, optimisation, trajectories

19. ABSTRACT

There are many diverse numerical methods that can be applied to solving path planning problems, however most of these are either not valid or impractical for solving anisotropic (direction-dependent) path planning problems. Ordered Upwind Methods (OUM) are a family of numerical methods for approximating the viscosity solution of static Hamilton-Jacobi-Bellman equations, and have been tailored to solve anisotropic optimal control problems.

There is little information in the literature regarding the implementation of OUM, and a wide range of computational techniques and meticulous algorithmic considerations are required to successfully implement OUM. A comprehensive, generic implementation of OUM is documented in this report, with the intention of minimising the technical barriers to employing OUM in real-world applications.